



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

*к лабораторной работе №5  
«Буферизованный и небуферизованный ввод-вывод»*

Студент: Батбилэг Н.

Группа: ИУ7-71Б

Оценка (баллы) \_\_\_\_\_

Преподаватель: Рязанова Н.Ю.

Москва.  
2021 г.

## Задачи: анализ особенностей работы функций ввода-вывода в UNIX/Linux

### 1. Общее задание

В лабораторной работе анализируется результат выполнения трех программ. Программы демонстрируют открытие одного и того же файла несколько раз. Реализация открытия файла в одной программе несколько раз выбрана для простоты. Такая ситуация возможна в системе, когда один и тот же файл несколько раз открывают разные процессы. Но для получения ситуаций аналогичных тем, которые демонстрируют приведенные программы надо было бы синхронизировать работу процессов. При выполнении асинхронных процессов такая ситуация вероятна и ее надо учитывать, чтобы избежать потери данных или получения неверного результата при выводе в файл. Проанализировать работу приведенных программ и объяснить результаты их работы.

### 2. Задание 1

В листинге 1 представлен код первого задания.

Листинг 1. test.c

```
1 #include <stdio.h>
2 #include <fcntl.h>
3
4 /*
5     На моей машине размер буфера 20 байт.
6     переведен в 12-символьный буфер.
7     По-видимому, 8 байтов было израсходовано
8     Библиотека stdio для бухгалтерского учета.
9 */
10
11 int main()
12 {
13     // иметь открытое соединение ядра с файлом алфавит.txt
14     int fd = open("alphabet.txt", O_RDONLY);
15
16
17     // создать два буферизованных потока ввода-вывода C,
18     // используя указанное выше соединение
19     FILE *fs1 = fdopen(fd, "r");
20     char buff1[20];
21     setvbuf(fs1, buff1, _IOFBF, 20);
22
23     FILE *fs2 = fdopen(fd, "r");
24     char buff2[20];
25     setvbuf(fs2, buff2, _IOFBF, 20);
26
27
28     // прочитать символ и записать его поочередно из fs1 и fs2
29     int flag1 = 1, flag2 = 2;
30     while (flag1 == 1 || flag2 == 1)
31     {
32         char c;
33         flag1 = fscanf(fs1, "%c", &c);
34         if (flag1 == 1) {
```

```

35     fprintf(stdout, "%c", c);
36 }
37 flag2 = fscanf(fs2, "%c", &c);
38 if (flag2 == 1) {
39     fprintf(stdout, "%c", c);
40 }
41 }
42 printf("\n");
43 return 0;
44 }

```

На рисунке 1 представлено содержимое файла alphabet.c и результат работы программы.

```

dalai@dalai-Inspiron-5567: ~/Desktop/BMSTU/labs/lab_05
dalai@dalai-Inspiron-5567:~/Desktop/BMSTU/labs/lab_05$ ls -a
.  alphabet.txt  .~lock.Отчет_лабораторная_5.odt#  Отчет_лабораторная_5.odt
.. a.out         test.c
dalai@dalai-Inspiron-5567:~/Desktop/BMSTU/labs/lab_05$ cat alphabet.txt
Abcdefghijklmnopqrstuvwxyz
dalai@dalai-Inspiron-5567:~/Desktop/BMSTU/labs/lab_05$ ./a.out
hubvcwdxeyfzg
ijklmnopqrst
dalai@dalai-Inspiron-5567:~/Desktop/BMSTU/labs/lab_05$ cat -A alphabet.txt
Abcdefghijklmnopqrstuvwxyz^M$
dalai@dalai-Inspiron-5567:~/Desktop/BMSTU/labs/lab_05$

```

Рисунок 1. Результаты работы программы.

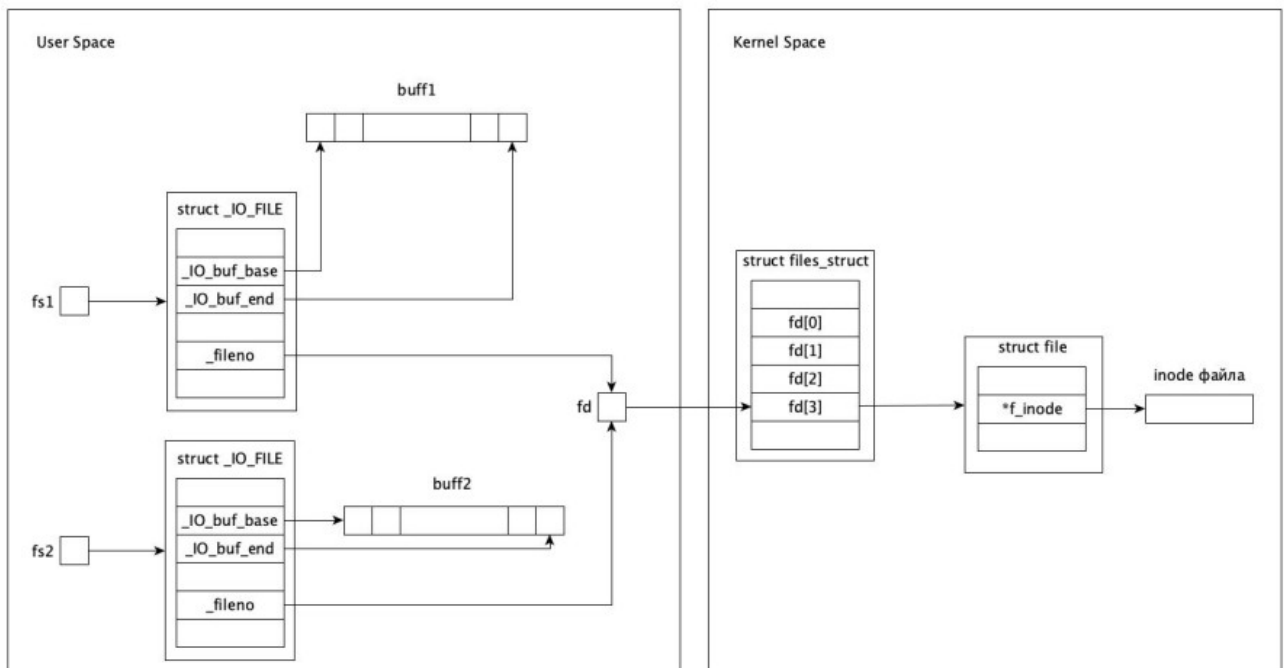


Рисунок 2. Связь дескрипторов в test.c.

Системный вызов `open()` создает дескриптор `fd` файла, открытого на чтение. Создается запись в системной таблице открытых файлов. Поток данных `fs1` и `fs2` связываются с

файловым дескриптором `fd` с помощью `fdopen()`. С помощью `setvbuf()` устанавливается блочная (полная) буферизация размером в 20 байт для каждого из потоков. Далее производится чтение из потоков (`fscanf()`) и вывод (`fprintf()`). В буфер `fs1` попадают первые 20 байт файла `alphabet.txt` (A...t), в буфер `fs2` – оставшиеся символы (u...z, \n). Символы выводятся по очереди из двух потоков. В потоках разное число символов, поэтому после вывода всех символов из второго потока выводятся символы только первого потока (после \n). Итог – “Aubvcwdxeyfzg\nhijklmnopqrst”. После этого отдельно выводится \n, чтобы вывод от программы не слился с последующим текстом в консоли.

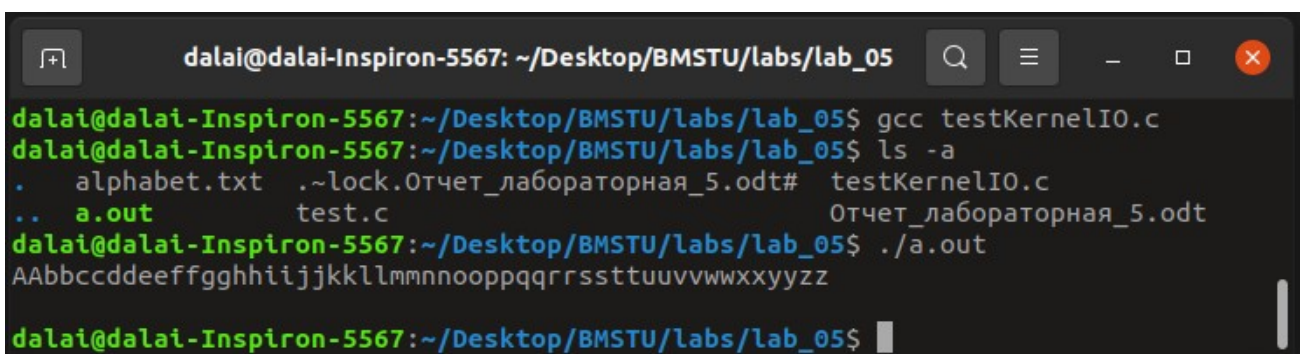
### 3. Задание 2

В листинге 2 представлен код задания.

Листинг 2. testKernelIO.c

```
1 #include <fcntl.h>
2 #include <unistd.h>
3 int main()
4 {
5     int flag1 = 1;
6     int flag2 = 1;
7     char c;
8     // Ядро открывает два подключения к файлу алфавит.txt
9     int fd1 = open("alphabet.txt", O_RDONLY);
10    int fd2 = open("alphabet.txt", O_RDONLY);
11    // читать символ и записывать его поочередно из соединений fs1 и fd2
12    while(flag1 == 1 && flag2 == 1)
13    {
14        flag1 = read(fd1, &c, 1);
15        if (flag1 == 1)
16        {
17            write(1, &c, 1);
18            flag2 = read(fd2, &c, 1);
19            if (flag2 == 1)
20                write(1, &c, 1);
21        }
22    }
23    return 0;
24 }
```

На рисунке 3 представлен результат работы программы.



```
dalai@dalai-Inspiron-5567: ~/Desktop/BMSTU/labs/lab_05
dalai@dalai-Inspiron-5567:~/Desktop/BMSTU/labs/lab_05$ gcc testKernelIO.c
dalai@dalai-Inspiron-5567:~/Desktop/BMSTU/labs/lab_05$ ls -a
.  alphabet.txt  .~lock.Отчет_лабораторная_5.odt#  testKernelIO.c
.. a.out          test.c                      Отчет_лабораторная_5.odt
dalai@dalai-Inspiron-5567:~/Desktop/BMSTU/labs/lab_05$ ./a.out
AAbbccddeeffgghhiijjkkllmmnnnooppqrrssttuuvvwxxyzz
dalai@dalai-Inspiron-5567:~/Desktop/BMSTU/labs/lab_05$
```

Рисунок 3. Результат работы программы.

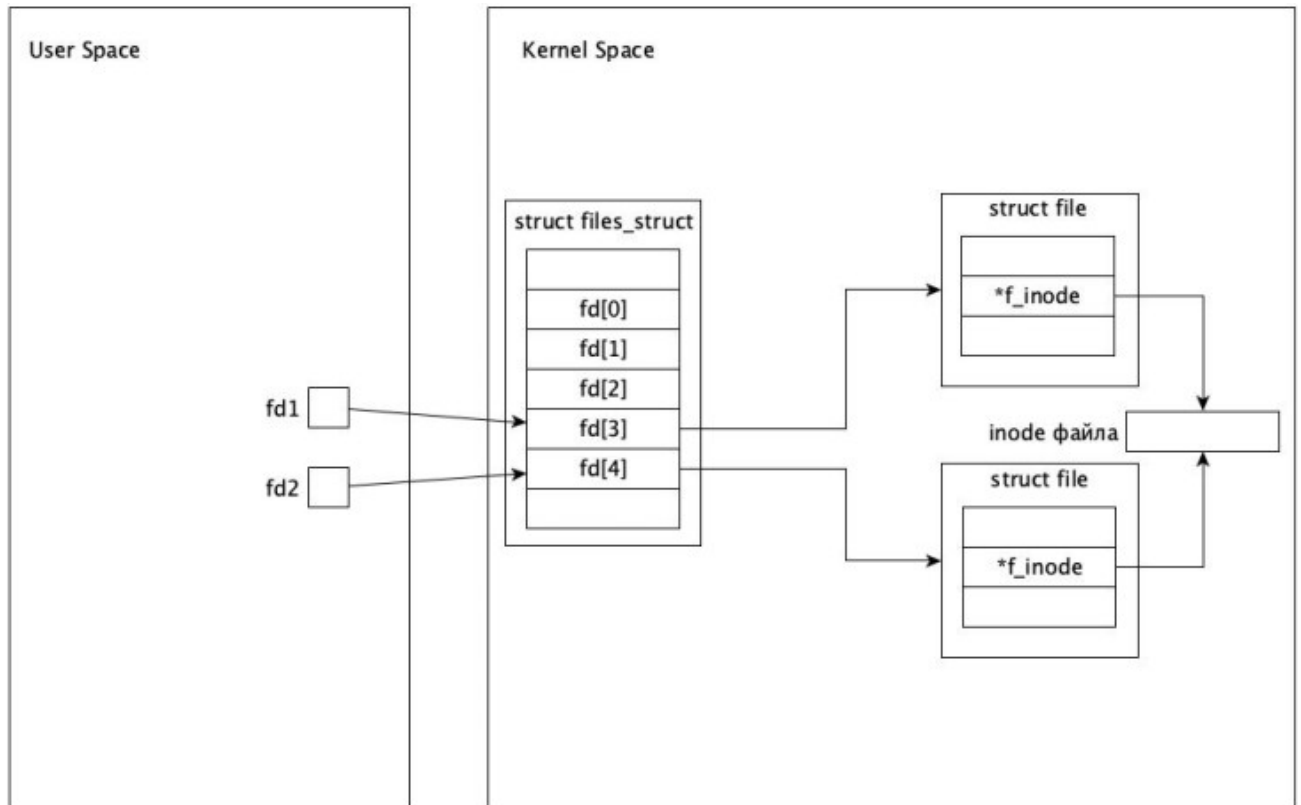


Рисунок 4. Связь дескрипторов в testKernel.

В данной программе создаются две разные записи в таблице открытых файлов и два дескриптора fd1 и fd2. Эти дескрипторы независимы друг от друга, поэтому все символы из файла выводятся на экран два раза – по одному из каждого дескриптора. Результат:

“Aabbccddeeffgghhiijjkkllmmnnnooppqrrssttuuvvwwxxyyzz\n\n”.

#### 4. Задание 3.

В листинге 3 представлен код задания 3.

Листинг 3. Test.c

```
1 #include <stdio.h>
2
3 int main() {
4     FILE *fd1 = fopen("out.txt", "w");
5     FILE *fd2 = fopen("out.txt", "w");
6
7     for (char c = 'a'; c <= 'z'; c++)
8     {
9         if (c % 2)
```

```

10         fprintf(fd1, "%c", c);
11     else
12         fprintf(fd2, "%c", c);
13 }
14
15 fclose(fd1);
16 fclose(fd2);
17 return 0;
18 }

```

На рисунке 5 представлен результат работы программы.

```

dalai@dalai-Inspiron-5567: ~/Desktop/BMSTU/labs/lab_05
dalai@dalai-Inspiron-5567:~/Desktop/BMSTU/labs/lab_05$ gcc Test.c
dalai@dalai-Inspiron-5567:~/Desktop/BMSTU/labs/lab_05$ ./a.out
dalai@dalai-Inspiron-5567:~/Desktop/BMSTU/labs/lab_05$ ls
alphabet.txt  out.txt  Test.c      Отчет_лабораторная_5.odt
a.out        test.c   testKernelIO.c
dalai@dalai-Inspiron-5567:~/Desktop/BMSTU/labs/lab_05$ cat out.txt
dalai@dalai-Inspiron-5567:~/Desktop/BMSTU/labs/lab_05$ cat out.txt
dalai@dalai-Inspiron-5567:~/Desktop/BMSTU/labs/lab_05$ cat out.txt
bdfhjlnprtvxzdalai@dalai-Inspiron-5567:~/Desktop/BMSTU/labs/lab_05$

```

Рисунок 5. Результат работы программы.

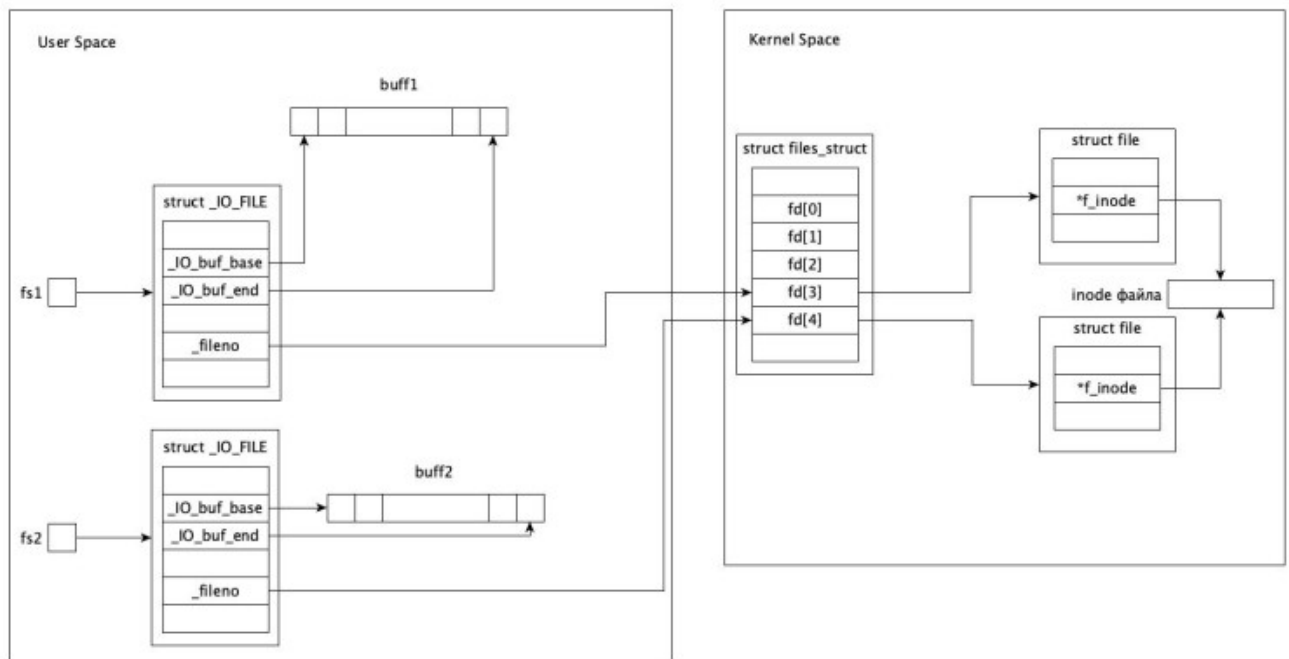


Рисунок 6. Связь дескрипторов в Test.c

Файл out.txt открывается дважды на запись с помощью fopen(), создаются два независимых дескриптора. Символы, имеющие нечетный код, записываются в fs1, четный – в fs2.

Функция `fprint()` обеспечивает буферизованный вывод — информация непосредственно записывается в файл только при вызове функции `fcolse()`, `fflush()`, либо при полном заполнении буфера.

Затем происходит вызов функции `fcolse()` для каждого буфера. Эта функция отделяет поток от файла. Если поток использовался для вывода, то все данные, содержащиеся в буфере принудительно запишутся в файл с использованием функции `fflush()`. Поскольку оба потока открыты на запись, то после выполнения второго `fcolse()`, данные, записанные в файл из первого потока, будут перезаписаны данными из второго. Поэтому в файле мы увидим буквы `b`, `d`, `f`, `h`, `j` и т.д.

## 5. Структура `FILE`

```
1 struct _IO_FILE
2 {
3     int _flags; /* Старшее слово - _IO_MAGIC, остальное - флаги. */
4
5     /* Следующие указатели соответствуют протоколу streambuf C ++. */
6     char *_IO_read_ptr; /* Текущий указатель чтения */
7
8     char *_IO_read_end; /* Конец области получения. */
9
10    char *_IO_read_base; /* Начало сдачи + получение площади. */
11
12    char *_IO_write_base; /* Начало положенного участка. */
13
14    char *_IO_write_ptr; /* Текущий указатель ввода. */
15
16    char *_IO_write_end; /* Конец положенной области. */
17
18    char *_IO_buf_base; /* Начало заповедной зоны. */
19
20    char *_IO_buf_end; /* Конец заповедной зоны. */
21
22    /* Следующие поля используются для поддержки резервного копирования и отмены. */
23    char *_IO_save_base; /* Указатель на начало нетекущей области получения. */
24    char *_IO_backup_base; /* Указатель на первый допустимый символ области */
25                          /* резервного копирования */
26    char *_IO_save_end; /* Указатель на конец нетекущей области получения. */
27    struct _IO_marker *_markers;
28    struct _IO_FILE *_chain;
29    int _fileno;
30    int _flags2;
31    __off_t _old_offset; /* Раньше это было _offset, но оно слишком мало. */
32    /* 1 + столбец номер фазы (); 0 неизвестно. */
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    };
```

## **6. Выводы**

Для ввода/вывода может использоваться буферизация, что ускоряет некоторые операции, связанные с вводом/выводом. Минус буферизации заключается в том, что нужно не забывать закрывать файлы после работы с ними.

Кроме того, при открытии одного и того же файла дважды создаются независимые записи о них, поэтому текущее положение в файле тоже независимо. Это можно привести к нежелательному исходу программы.

Данные могут повторяться или теряться — в зависимости от операции, что и продемонстрировано в данной лабораторной работе.