**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»

# Лабораторная работа №5

**Дисциплина** Операционные системы

**Тема** Буферизованный и небуферизованный ввод-вывод

**Студент** Набиев Ф.М.

**Группа** ИУ7–63Б

**Оценка (баллы)** _____

**Преподаватель** Рязанова Н.Ю.

*Москва, 2020 г.*

## 1 Задание №1

В листинге 1.1 приведён текст первой программы.

Листинг 1.1 – Текст первой программы

```
1  #include <stdio.h>
2  #include <fcntl.h>
3  #include <errno.h>
4  #include <unistd.h>
5  #include <stdlib.h>
6  #include <string.h>
7
8  int main(void)
9  {
10     char c;
11     char buff1[20];
12     char buff2[20];
13     int flag1, flag2;
14
15     int fd = open("alphabet.txt", O_RDONLY);
16     if (fd == -1)
17     {
18         fprintf(stderr, "%s: %s\n", "open alphabet.txt", strerror(errno));
19         exit(1);
20     }
21
22     FILE *fs1 = fdopen(fd, "r");
23     if (!fs1)
24     {
25         fprintf(stderr, "first fdopen %d: %s\n", fd, strerror(errno));
26         exit(1);
27     }
28     setvbuf(fs1, buff1, _IOFBF, 20);
29     FILE *fs2 = fdopen(fd, "r");
30     if (!fs2)
31     {
32         fprintf(stderr, "second fdopen %d: %s\n", fd, strerror(errno));
33         exit(1);
34     }
35     setvbuf(fs2, buff2, _IOFBF, 20);
36
37     flag1 = flag2 = 1;
38     while(flag1 == 1 || flag2 == 1)
39     {
40         flag1 = fscanf(fs1, "%c", &c);
41         if (flag1 == 1)
42             fprintf(stdout, "%c", c);
```

```
43
44          flag2 = fscanf(fs2, "%c", &c);
45          if (flag2 == 1)
46              fprintf(stdout, "%c", c);
47      }
48
49      close(fd);
50
51      return 0;
52  }
```

На рисунке 1.1 изображён результат работы этой программы.

```
$ ./a.out && echo '\n'
Aubvcwdxeyfzghijklmnopqrst
```

Рис. 1.1 – Результат работы первой программы

В рассматриваемой программе в результате вызова системного вызова open создается новый дескриптор файла, который открывается только на чтение. Так же создается запись в системной таблице открытых файлов. Текущая позиция устанавливается на начало файла.

Далее функция fdopen в результате двух вызовов создает два разных объекта структуры FILE — два потока ввода, ссылающихся на один ранее созданный файловый дескриптор, а функция setvbuf устанавливает блочную буферизацию размером 20 байт для этих потоков.

При первом вызове fscanf, то есть при первой операции чтения из файла, происходит заполнения буфера содержимым файла до тех пор, пока в нем есть место или пока не будет достигнут конец файла. Так как оба потока ссылаются на один и тот же файловый дескриптор (а значит и на одну и ту же запись в системной таблице открытых файлов) и размер каждого буфера равен 20 байт, то в первый в порядке чтения буфер будет записано первые 20 символов файла, в во второй — оставшиеся 6.

Таким образом, на экран выводится строка, состоящая из символов, поочередно выведенных из первого и из второго буферов.

На рисунке 1.2 изображена связь между созданными дескрипторами.

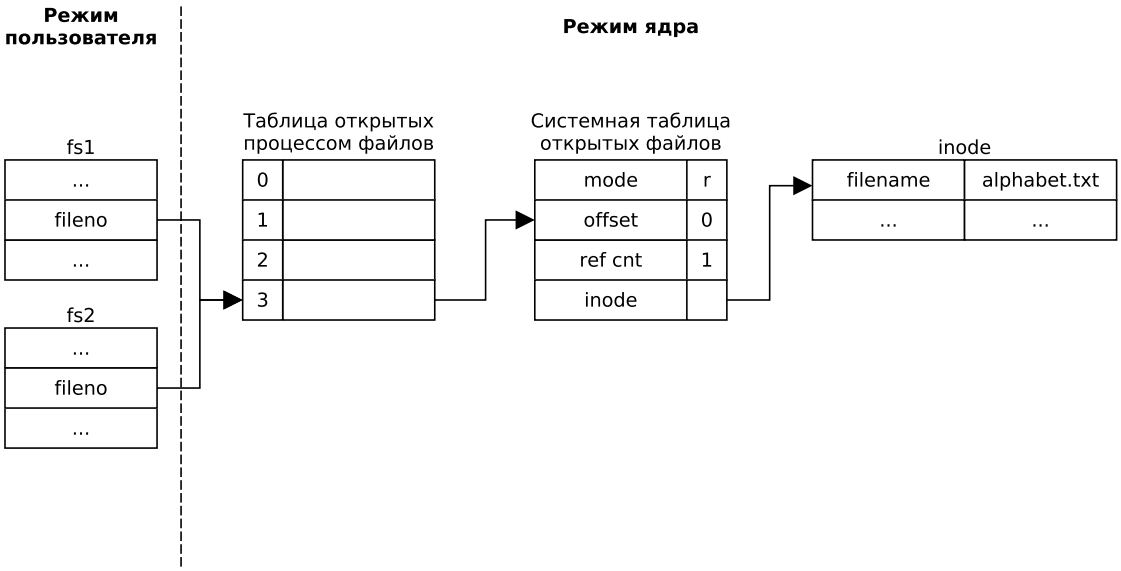


Рис. 1.2 – Связь между дескрипторами

## 2 Задание №2

В листинге 2.1 приведён текст второй программы.

Листинг 2.1 – Текст второй программы

```
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
    char c;
    int flag1, flag2;

    int fd1 = open("alphabet.txt", O_RDONLY);
    if (fd1 == -1)
    {
        fprintf(stderr, "%s: %s\n", "first open alphabet.txt",
                strerror(errno));
        exit(1);
    }
    int fd2 = open("alphabet.txt", O_RDONLY);
    if (fd2 == -1)
    {
        fprintf(stderr, "%s: %s\n", "second open alphabet.txt",
                strerror(errno));
        exit(1);
    }

    do
    {
        if ((flag1 = read(fd1, &c, 1)))
            write(1, &c, 1);

        if ((flag2 = read(fd2, &c, 1)))
            write(1, &c, 1);
    }
    while (flag1 || flag2);

    close(fd1);
    close(fd2);

    return 0;
}
```

На рисунке 2.1 изображён результат работы второй программы.

```
$ ./a.out && echo '\n'
AAbbccddeeffgghhiijjkkllmmnnooppqqrrssttuuvvwwxxyyzz
```

Рис. 2.1 – Результат работы второй программы

В результате двух последовательных вызовов open создаются два разных дескриптора одного файла, который оба раза открывается на чтение, и две записи в системной таблице открытых файлов. Каждому дескриптору соответствует своя текущая позиция в файле.

Таким образом получается, что на каждой итерации цикла два раза происходит чтение одного и того же символа из файла при помощи разных файловых дескрипторов. Следовательно, на каждой итерации цикла выводится два одинаковых символа.

На рисунке 2.2 изображена связь между созданными дескрипторами.

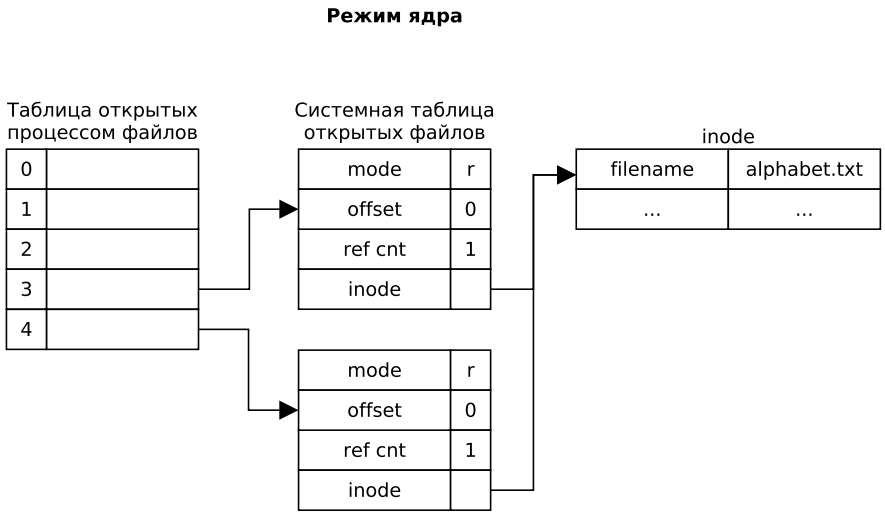


Рис. 2.2 – Связь между дескрипторами

## 3 Задание №3

В листинге 3.1 приведён текст третьей программы.

Листинг 3.1 – Текст третьей программы

```
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
    FILE *fss[2];
    const char alphabet[] = "Abcdefghijklmnopqrstuvwxyz";

    fss[0] = fopen("out.txt", "wr");
    if (!fss[0])
    {
        fprintf(stderr, "first fopen: %s\n", strerror(errno));
        exit(1);
    }
    fss[1] = fopen("out.txt", "wr");
    if (!fss[1])
    {
        fprintf(stderr, "second fopen: %s\n", strerror(errno));
        exit(1);
    }

    for (int i = 0; i < sizeof(alphabet)-1; ++i)
        fprintf(fss[i % 2], "%c", alphabet[i]);

    fclose(fss[0]);
    fclose(fss[1]);

    return 0;
}
```

На рисунке 3.1 изображён результат работы третьей программы.

```
$ ./a.out
$ cat out.txt && echo '\n'
bdfhjlnprtvxz
```

Рис. 3.1 – Результат работы третьей программы

В результате двух последовательных вызовов fopen создаётся два

потока на запись. Эти потоки ссылаются на различные файловые дескрипторы, а значит они имеют разные указатели на текущую позицию в файле.

Так как fopen создаёт поток, ввод-вывод для которого выполняется с буферизацией, запись непосредственно в файл осуществляется только при вызове функции fclose, fflush, либо при полном заполнении буфера.

На каждой итерации цикла символы с чётными индексами записываются в буфер первого потока, а с нечётными — второго.

Далее происходят два последовательных вызова fclose для обоих потоков. Так как данные потоки были открыты на запись, эта функция принудительно запишет данные из буфера соответствующего потока в файл, используя функцию fflush. При этом данные, записанные файл из второго потока, перезапишут данные, записанные в файл из первого.

На рисунке 3.2 изображена связь между созданными дескрипторами.

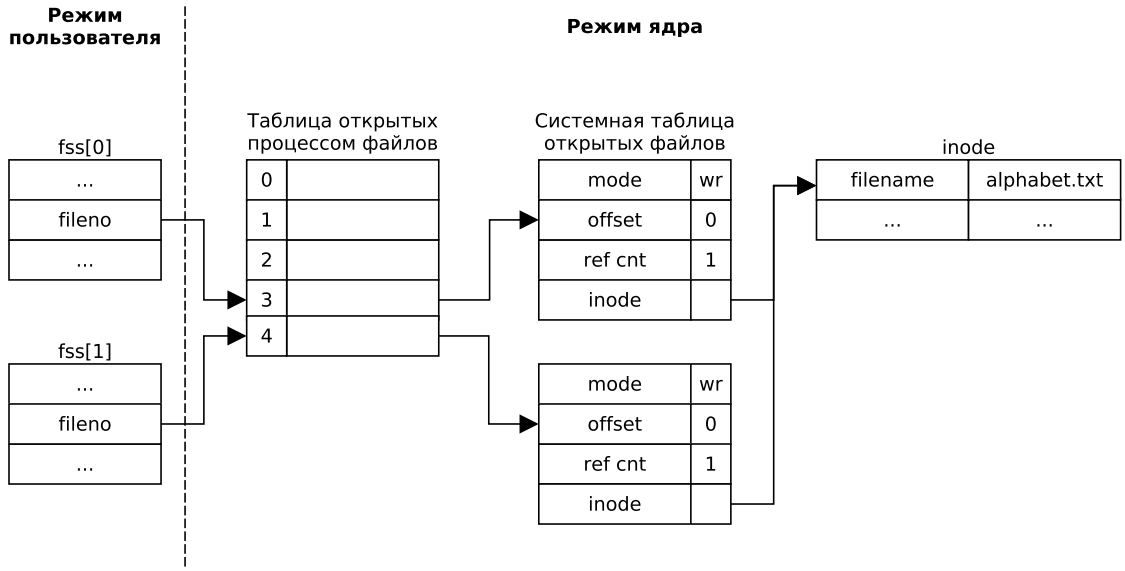


Рис. 3.2 – Связь между дескрипторами

## 4 Структура FILE

В листинге 4.1 приведено определение структуры FILE.

Листинг 4.1 – Структура FILE

```
1  // /usr/include/x86_64-linux-gnu/bits/types/FILE.h
2  #ifndef __FILE_defined
3  #define __FILE_defined 1
4
5  struct _IO_FILE;
6
7  /* The opaque type of streams.  This is the definition used elsewhere.  */
8  typedef struct _IO_FILE FILE;
9
10 #endif
11
12 // /usr/include/x86_64-linux-gnu/bits/types/struct_FILE.h
13 /* Copyright (C) 1991-2018 Free Software Foundation, Inc.
14    This file is part of the GNU C Library.
15
16    The GNU C Library is free software; you can redistribute it and/or
17    modify it under the terms of the GNU Lesser General Public
18    License as published by the Free Software Foundation; either
19    version 2.1 of the License, or (at your option) any later version.
20
21    The GNU C Library is distributed in the hope that it will be useful,
22    but WITHOUT ANY WARRANTY; without even the implied warranty of
23    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
24    Lesser General Public License for more details.
25
26    You should have received a copy of the GNU Lesser General Public
27    License along with the GNU C Library; if not, see
28    <http://www.gnu.org/licenses/>.  */
29
30 #ifndef __struct_FILE_defined
31 #define __struct_FILE_defined 1
32 // ...
33 /* The tag name of this struct is _IO_FILE to preserve historic
34    C++ mangled names for functions taking FILE* arguments.
35    That name should not be used in new code.  */
36 struct _IO_FILE
37 {
38   int _flags;       /* High-order word is _IO_MAGIC; rest is flags. */
39
40   /* The following pointers correspond to the C++ streambuf protocol. */
41   char *_IO_read_ptr;   /* Current read pointer */
42   char *_IO_read_end;   /* End of get area. */
```

```
43    char *_IO_read_base;  /* Start of putback+get area. */
44    char *_IO_write_base; /* Start of put area. */
45    char *_IO_write_ptr;  /* Current put pointer. */
46    char *_IO_write_end;  /* End of put area. */
47    char *_IO_buf_base;   /* Start of reserve area. */
48    char *_IO_buf_end;    /* End of reserve area. */
49
50    /* The following fields are used to support backing up and undo. */
51    char *_IO_save_base; /* Pointer to start of non-current get area. */
52    char *_IO_backup_base;
53    /* Pointer to first valid character of backup area */
54    char *_IO_save_end; /* Pointer to end of non-current get area. */
55
56    struct _IO_marker *_markers;
57
58    struct _IO_FILE *_chain;
59
60    int _fileno;
61    int _flags2;
62    __off_t _old_offset; /* This used to be _offset but it's too small.  */
63
64    /* 1+column number of pbase(); 0 is unknown. */
65    unsigned short _cur_column;
66    signed char _vtable_offset;
67    char _shortbuf[1];
68
69    _IO_lock_t *_lock;
70 #ifdef _IO_USE_OLD_IO_FILE
71 };
72
73 struct _IO_FILE_complete
74 {
75    struct _IO_FILE _file;
76 #endif
77    __off64_t _offset;
78    /* Wide character stream stuff.  */
79    struct _IO_codecvt *_codecvt;
80    struct _IO_wide_data *_wide_data;
81    struct _IO_FILE *_freeres_list;
82    void *_freeres_buf;
83    size_t __pad5;
84    int _mode;
85    /* Make sure we don't get into trouble again.  */
86    char _unused2[15 * sizeof (int) - 4 * sizeof (void *) - sizeof (size_t)];
87 };
88 // ...
89 #endif
```