

Федеральное государственное бюджетное образовательное учреждение высшего  
профессионального образования



**«Московский государственный технический  
университет имени Н.Э. Баумана»  
(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»**

Студент группы ИУ7-63Б: М. В. Заколесник  
(Фамилия И.О.)

Преподаватель: Н. Ю. Рязанова  
(Фамилия И.О.)

# 1. CIO.c

## Текст программы

```
1 //testCIO.c
2 #include <stdio.h>
3 #include <fcntl.h>
4
5 /*
6  On my machine, a buffer size of 20 bytes
7  translated into a 12-character buffer.
8  Apparently 8 bytes were used up by the
9  stdio library for bookkeeping.
10 */
11
12 int main()
13 {
14     // have kernel open connection to file alphabet.txt
15     int fd = open("alphabet.txt", O_RDONLY);
16
17     // create two a C I/O buffered streams using the above connection
18     // associates a stream with the existing file descriptor
19     FILE *fs1 = fdopen(fd, "r");
20     char buff1[20];
21     setvbuf(fs1, buff1, _IOFBF, 20); // set fully buffering
22
23     FILE *fs2 = fdopen(fd, "r");
24     char buff2[20];
25     setvbuf(fs2, buff2, _IOFBF, 20);
26
27     // read a char and write it alternately from fs1 and fs2
28     int flag1 = 1, flag2 = 2;
29     while(flag1 == 1 || flag2 == 1)
30     {
31         char c;
32         flag1 = fscanf(fs1, "%c", &c);
33
34         if (flag1 == 1) { fprintf(stdout, "%c", c); }
35         flag2 = fscanf(fs2, "%c", &c);
36         if (flag2 == 1) { fprintf(stdout, "%c", c); }
37     }
38
39     return 0;
40 }
```

## Вывод

1	aubvcwdxeyfzg
2	hijklmnopqrst

## Анализ

Анализ работы программы и результатов: в самом начале программы создается дескриптор файла “alphabet.txt” в таблице файловых дескрипторов процесса и запись в таблице открытых файлов. В записи этой таблицы хранится текущее смещение указателя в файле, которое используется во всех операциях чтения и записи в файл, а также режим открытия файла (O\_RDONLY, O\_WRONLY или O\_RDWR). При выполнении операции чтения-записи система выполняет неявный сдвиг указателя. Далее создаются два объекта типа FILE которые ссылаются на созданный файловый дескриптор и при помощи функции *setvbuf(...)* мы задаем размер буфера каждого объекта равный 20 символам. При первом вызове *fscanf(fs1, "%c &c)*, буфер первого файлового дескриптора предварительно заполняется первыми 20 символами - abcdefghijklmnopqrst, а при втором *fscanf(fs2, "%c &c)* буфер второго файлового дескриптора заполняется оставшейся частью – uvwxyz (т. к. обе структуры FILE ссылаются на одну и ту же запись в таблице открытых файлов, а при первом чтении указатель в файле уже сместился на 20 позиций). Далее в цикле происходит поочередная печать по одному символу, который уже берется не из файла, а из соответствующего буфера файлового дескриптора. Поэтому мы можем наблюдать такие результаты.

## 2. testKernelIO.c

### Текст программы

```
1 //testKernelIO.c
2 #include <fcntl.h>
3
4 int main()
5 {
6     // have kernel open two connection to file alphabet.txt
7     int fd1 = open("alphabet.txt",O_RDONLY);
8     int fd2 = open("alphabet.txt",O_RDONLY);
9
10
11     // read a char & write it alternatingly from connections fs1 & fd2
12     while(1)
13     {
14         char c;
15         if (read(fd1,&c,1) != 1) break;
16         write(1,&c,1);
17         if (read(fd2,&c,1) != 1) break;
18         write(1,&c,1);
19     }
20
21     return 0;
22 }
```

### Вывод

```
1 aabbccddeeffghhijjkkllmmnnnooppqrrssttuuvvwwxxyyzz
```

### Анализ

В данной программе мы создали два дескриптора файла “alphabet.txt” и две различные записи в таблице открытых файлов системы. В этом случае, каждая запись имеет своё независимое смещение в файле. Поэтому при поочередной печати символа в цикле мы имеем, что каждый указатель смещается независимо друг от друга.

### 3. FOpen.c

```
1 #include <stdio .h>
2
3 int main()
4 {
5     FILE *fd [2];
6
7     fd [0] = fopen ("FOpen_output.txt", "w");
8     fd [1] = fopen ("FOpen_output.txt", "w");
9
10    int curr = 0;
11    for(char c = 'a'; c <= 'z'; c++, curr = ((curr != 0) ? 0 : 1))
12    {
13        fprintf(fd[curr], "%c", c);
14
15    }
16
17    fclose(fd[0]);
18    fclose(fd[1]);
19
20    return 0;
21 }
```

#### Вывод

```
1 bdfhjlnprtvxz
```

#### Анализ

при вызове функции `fopen()` создаются 2 файловых дескриптора и 2 записи в таблице открытых файлов. Вспомним, что при вызове `fprintf(...)`, запись производится в буфер. И только тогда, когда буфер будет заполнен полностью или если, будут вызваны функции `fclose(...)`, `fflush(...)` данные будут записаны в файл. По этой причине, когда в программе вызвалась функция `fclose(fs1)`, в файл записались все буквы английского алфавита, которые были в буфере файлового дескриптора `fs1` (acegikmoqsuwу), тогда как после вызова функции `fclose(fs2)`, содержимое файла удалилось (т.к. мы открыли файл для записи с режимом “w”), и записалась информация из буфера `fs2`.

#### 4. Схемы программ

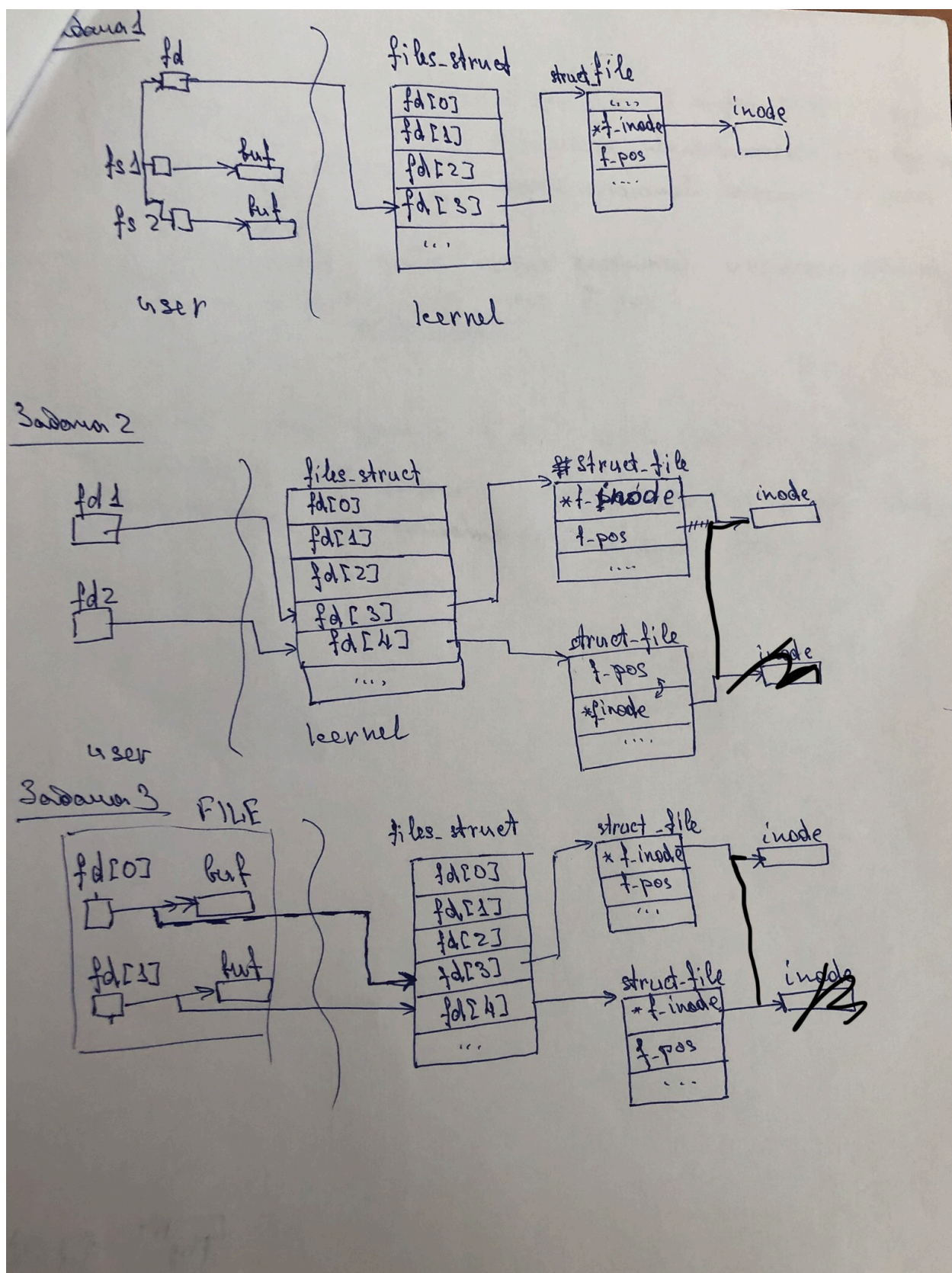


Рис. 1: Схемы 1-3

## 5. Структура FILE

Листинг 1: Структура FILE для OS X.

```
1 typedef struct __sFILE {
2     unsigned char *_p; /* current position in (some) buffer */
3     int _r; /* read space left for getc() */
4     int _w; /* write space left for putc() */
5     short _flags; /* flags, below; this FILE is free if 0 */
6     short _file; /* fileno, if Unix descriptor, else -1 */
7     struct __sbuf _bf; /* the buffer (at least 1 byte, if !NULL) */
8     int _lbfdsize; /* 0 or -_bf._size, for inline putc */
9
10    /* operations */
11    void *_cookie; /* cookie passed to io functions */
12    int (* __Nullable __close)(void *);
13    int (* __Nullable __read)(void *, char *, int);
14    fpos_t (* __Nullable __seek)(void *, fpos_t, int);
15    int (* __Nullable __write)(void *, const char *, int);
16
17    /* separate buffer for long sequences of ungetc() */
18    struct __sbuf _ub; /* ungetc buffer */
19    struct __sFILEX *_extra; /* additions to FILE to not break ABI */
20    int _ur; /* saved _r when _r is counting ungetc data */
21
22    /* tricks to meet minimum requirements even when malloc() fails */
23    unsigned char _ubuf[3]; /* guarantee an ungetc() buffer */
24    unsigned char _nbuf[1]; /* guarantee a getc() buffer */
25
26    /* separate buffer for fgetln() when line crosses buffer boundary */
27    struct __sbuf _lb; /* buffer for fgetln() */
28
29    /* Unix stdio files get aligned to block boundaries on fseek() */
30    int _blksize; /* stat.st_blksize (may be != _bf._size) */
31    fpos_t _offset; /* current lseek offset (see WARNING) */
32 } FILE;
```

## 6. Вывод

Исходя из вышеприведенных рассуждений, можно сделать несколько выводов.

- 1) При буфферизованном вводе/выводе необходимо учитывать факт записи/чтения данных из буфера, т.к. неправильные действия с данными, записываемыми(или считываемыми) в(из) файл(а), могут привести к неправильной последовательности данных(пример 1) или даже к их потере(пример 3).
- 2) При небуфферизованном вводе/выводе необходимо учитывать, что при одновременном открытии одного и того же файла создается дескриптор открытого файла(столько дескрипторов, сколько раз был открыт файл). Каждый дескриптор struct \_file имеет поле f\_pos, указывающий на позицию чтения или записи в логическом файле.