



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего 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»

---

## Лабораторная работа № 5

Дисциплина	Операционные системы.
Тема	Буферизованный и не буферизованный ввод-вывод.
Студент	Сиденко А.Г.
Группа	ИУ7-63Б
Оценка (баллы)	
Преподаватель	Рязанова Н.Ю.

Москва, 2020 г.

**Задание 1:** Проанализировать работу приведенной программы и объяснить результат ее работы.

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <errno.h>
4 #include <string.h>
5 #include <unistd.h>
6
7 int main()
8 {
9     // Создает файловый дескриптор для открытого файла
10    int fd = open("alphabet.txt", O_RDONLY);
11    if (fd == -1)
12    {
13        printf("%s", strerror(errno));
14        return errno;
15    }
16
17    // Создает 2 буферизованных потока ввода-вывода,
18    // используя файловый дескриптор
19    FILE *fs1 = fdopen(fd, "r");
20    if (fs1 == NULL)
21    {
22        printf("%s", strerror(errno));
23        return errno;
24    }
25    char buff1[20];
26    setvbuf(fs1, buff1, _IOFBF, 20);
27
28    FILE *fs2 = fdopen(fd, "r");
29    if (fs2 == NULL)
30    {
31        printf("%s", strerror(errno));
32        return errno;
33    }
34    char buff2[20];
35    setvbuf(fs2, buff2, _IOFBF, 20);
36
37    // Чтение char и вывод на экран попеременно из fs1 и fs2
38    int flag1 = 1, flag2 = 2;
39    while (flag1 == 1 || flag2 == 1)
40    {
41        char c;
42        flag1 = fscanf(fs1, "%c", &c);
43        if (flag1 == 1)
44        {
45            fprintf(stdout, "%c", c);
46        }
47
48        flag2 = fscanf(fs2, "%c", &c);
```

```

49     if (flag2 == 1)
50     {
51         fprintf(stdout, "%c", c);
52     }
53 }
54
55 close(fd);
56 return 0;
57 }

```

### Результат:

```

+ Operating_systems/semestr2/lab5 semestr2_lab5± ./testCI0.exe
Aubvcwdxeyfzghijklmnopqrst%

```

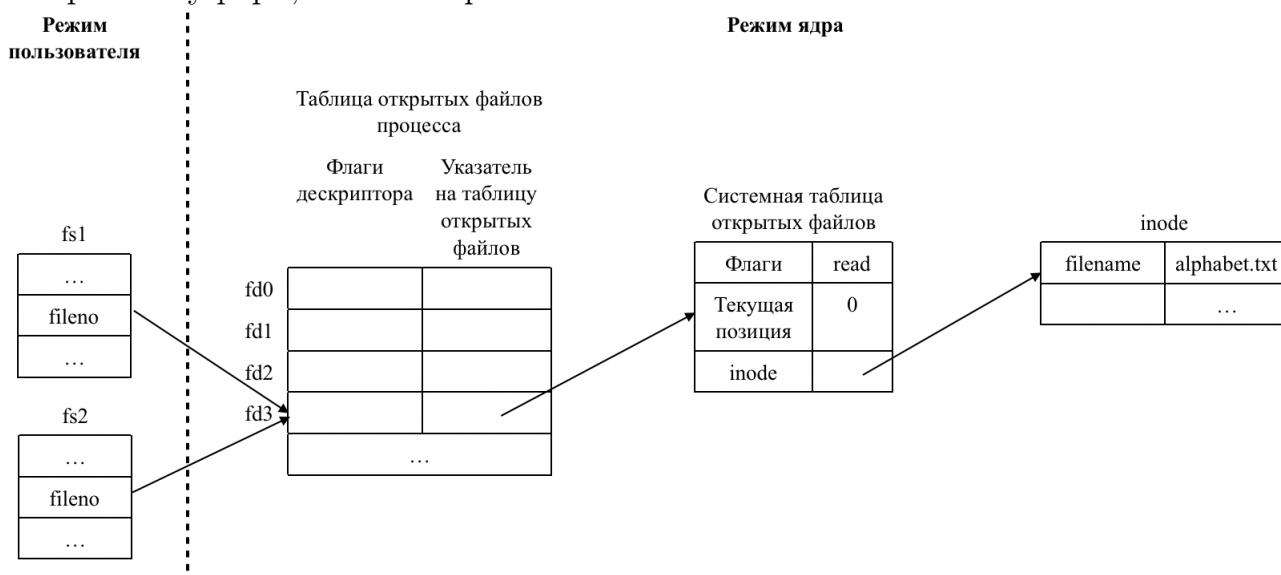
При использовании системного вызова `open()` создается новый файловый дескриптор для открытого файла, запись в общесистемной таблице открытых файлов. Текущая позиция устанавливается на начало файла. Далее системный вызов `fdopen()` создает два разных объекта структуры `FILE`, которые ссылаются на один файловый дескриптор `fd`, `setvbuf()` устанавливает тип буферизации блоком размером 20 байт.

Когда первая операция ввода-вывода выполняется над файлом, получается буфер:

При первом вызове `fscanf()` буфер ввода структуры `FILE` заполняется либо до конца буфера, либо до конца файла. Поэтому буфер заполняется полностью первыми 20 символами и значение текущей позиции смещается.

Так как `fs1` и `fs2` ссылаются на одну и ту же запись в системной таблице открытых файлов (значение позиции файла одинаковое), то при следующем вызове `fscanf()` буфер ввода структуры `fs2` считывает последние 6 символов из файла и символ конца файла.

Тогда результатом будет являться строка, где символы поочередно выводятся то из первого буфера, то из второго.



**Задание 2:** Проанализировать работу приведенной программы и объяснить результат ее работы.

```
1 #include <fcntl.h>
2 #include <unistd.h>
3 #include <errno.h>
4 #include <string.h>
5 #include <stdio.h>
6
7 int main()
8 {
9     // Создает файловый дескриптор для открытого файла 2 раза
10    int fd1 = open("alphabet.txt", O_RDONLY);
11    if (fd1 == -1)
12    {
13        printf("%s", strerror(errno));
14        return errno;
15    }
16
17    int fd2 = open("alphabet.txt", O_RDONLY);
18    if (fd2 == -1)
19    {
20        printf("%s", strerror(errno));
21        return errno;
22    }
23
24    // Чтение char и вывод на экран попеременно
25    char c1;
26    char c2;
27    // read возвращает количество действительно считанных байт.
28    // Соответственно, когда кол-во считанных байтов не будет равно 1,
29    // значит или конец файла или ошибка.
30    while ((read(fd1, &c1, 1) == 1) && (read(fd2, &c2, 1) == 1))
31    {
32        write(1, &c1, 1);
33        write(1, &c2, 1);
34    }
35
36    close(fd1);
37    close(fd2);
38
39    return 0;
40 }
```

**Результат:**



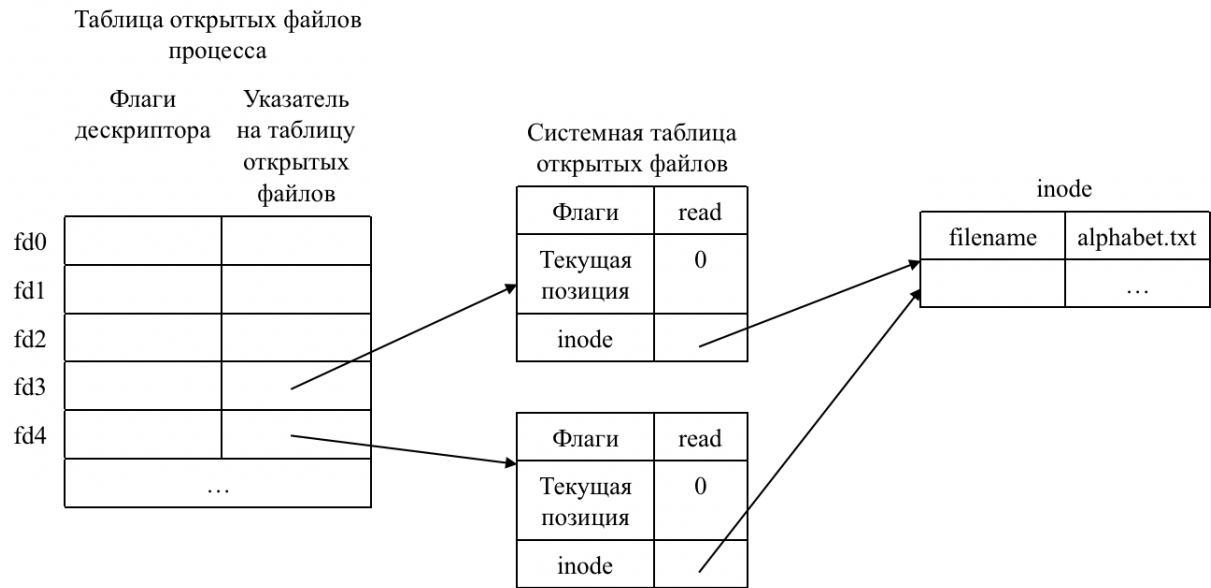
При использовании системного вызова `open()` создаются 2 разных файловых дескриптора для открытого файла, 2 записи в общесистемной таблице открытых файлов.

Файловые дескрипторы разные, поэтому у каждого своя текущая позиция

файла.

Тогда с помощью системных вызовов read(), write() в результате получится строка с дублирующимися символами.


**Режим ядра**



**Задание 3:** Написать программу, которая открывает один и тот же файл два раза с использованием библиотечной функции `fopen()`. Для этого объявляются два файловых дескриптора. В цикле записать в файл буквы латинского алфавита поочередно передавая функции `fprintf()` то первый дескриптор, то – второй. Результат прокомментировать.

```
1 #include <fcntl.h>
2 #include <stdio.h>
3 #include <errno.h>
4 #include <string.h>
5
6 int main()
7 {
8     // Открывает файл и связывает с ним буферизованный поток
9     FILE *fd1 = fopen("test3.txt", "w");
10    if (fd1 == NULL)
11    {
12        printf("%s", strerror(errno));
13        return errno;
14    }
15
16    FILE *fd2 = fopen("test3.txt", "w");
17    if (fd1 == NULL)
18    {
19        printf("%s", strerror(errno));
20        return errno;
21    }
22
23    // Цикл по всем буквам алфавита (нечетные – fd1, четные – fd2)
24    for(char c = 'a'; c <= 'z'; c++)
25    {
26        if (c % 2)
27        {
28            fprintf(fd1, "%c", c);
29        }
30        else
31        {
32            fprintf(fd2, "%c", c);
33        }
34    }
35
36    fclose(fd1);
37    fclose(fd2);
38    return 0;
39 }
```

**Результат:**



```
Operating_systems/semestr2/lab5 ➤ ./test3.exe
Operating_systems/semestr2/lab5 ➤ cat test3.txt
bdfhjlnprtvxz%
```

При вызове функции `fopen()` 2 раза с аргументом «w» (открытие для записи)

создаются 2 разных файловых дескриптора файла и две независимые позиции в файле указывают на начало.

В цикле с использованием функции `fprintf` в буферизованные потоки записываются буквы от а до z, нечетные в первый поток (а, с, е...), а четные во второй (b, d, f ...). Поскольку используются различные дескрипторы, то смещение текущей позиции файла при каждом вызове `fprintf()` происходит независимо.

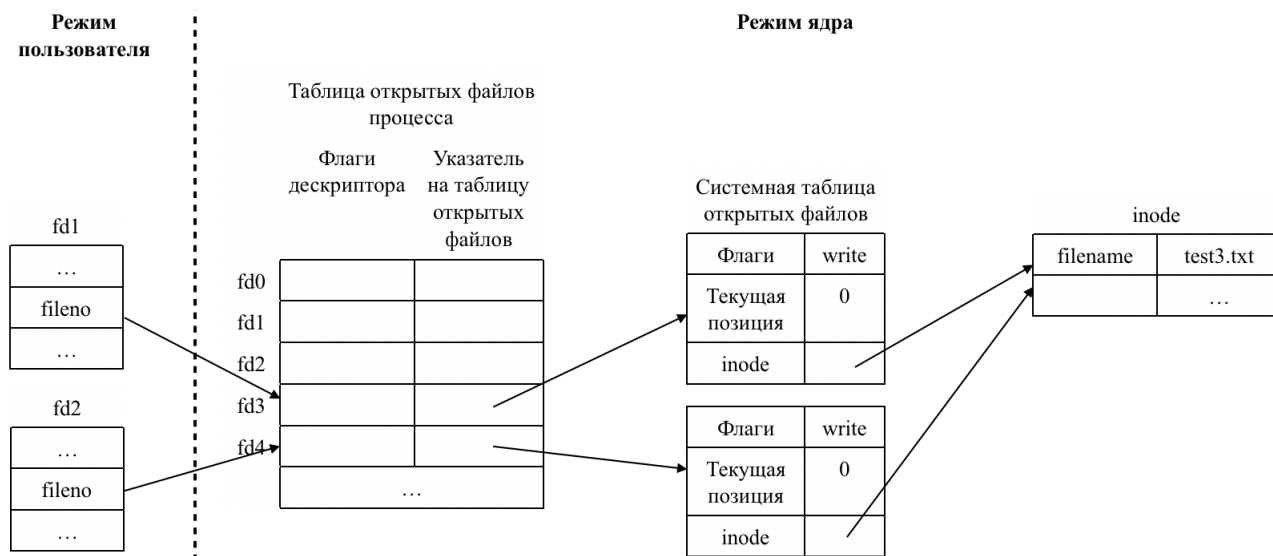
Так как `fprintf()` обеспечивает буферизованный вывод – запись непосредственно в файл осуществляется только при вызове функции `fclose()`, `fflush()`, либо при полном заполнении буфера.

Функция `fclose(fd1)` очистит поток, на который указывает `fd1` (запись любых буферизованных выходных данных с помощью `fflush` (принудительно записывает все буферизированные данные в устройство вывода данных)), и закроет файловый дескриптор.

Далее вызывается `fclose(fd2)`, так как оба потока открыты открыты на запись в файл, то после его выполнения данные в файле, записанные с помощью первого потока, перезапишутся данными из второго потока.

Если хотим, чтобы в файл можно было добавлять данные с помощью двух независимых дескрипторов (без опасения нарушить вывод), то необходимо открыть файл в режиме добавления с аргументом «а», тогда все последующие операции записи в файл будут работать с current end-of-file.

Информация будет записана в файл в том порядке, в котором процессы записывали ее в файл (запись непосредственно в файл осуществляется только при вызове функции `fclose()`, `fflush()`, либо при полном заполнении буфера).



## struct FILE

```
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 _lbfsz;           /* 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;
```

Функция `fileno(FILE *stream)` возвращает целочисленный файловый дескриптор, связанный с потоком, на который указывает `stream`.