



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  
по операционным системам**

Студент Колганов О.С.

Группа ИУ7 — 62Б

Преподаватель Рязанова Н.Ю.

Москва.  
2020 г.

## Программа 1

Код программы:

```
#include <stdio.h>
#include <fcntl.h>

int main()
{
    // have kernel open connection to file alphabet.txt
    int fd = open("alphabet.txt", O_RDONLY);

    // create two a C I/O buffered streams using the above connection
    FILE* fs1 = fdopen(fd, "r");
    char buff1[20];
    setvbuf(fs1, buff1, _IOFBF, 20);

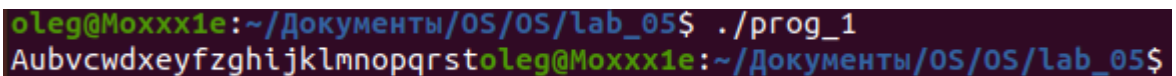
    FILE* fs2 = fdopen(fd, "r");
    char buff2[20];
    setvbuf(fs2, buff2, _IOFBF, 20);

    // read a char & write it alternatingly from fs1 and fs2
    int flag1 = 1, flag2 = 2;

    while (flag1 == 1 || flag2 == 1) {
        char c;
        flag1 = fscanf(fs1, "%c", &c);
        if (flag1 == 1) {
            fprintf(stdout, "%c", c);
        }
        flag2 = fscanf(fs2, "%c", &c);
        if (flag2 == 1) {
            fprintf(stdout, "%c", c);
        }
    }

    return 0;
}
```

Результат работы программы:



```
oleg@Moxxx1e:~/Документы/OS/OS/lab_05$ ./prog_1
Aubvcwdxeyfzghijklmnopqrstoleg@Moxxx1e:~/Документы/OS/OS/lab_05$
```

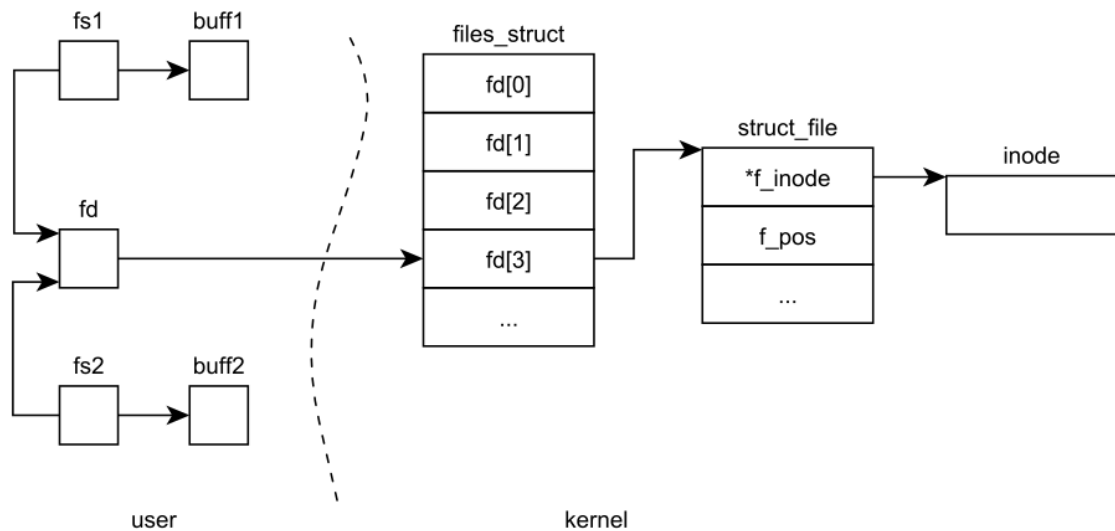
Анализ полученного результата:

В данной программе системный вызов `open()` открывает файл на чтение (так как передан флаг `O_RDONLY`) и возвращает файловый дескриптор `fd`, при этом указатель устанавливается в начало файла. В результате вызова появляется новый открытый файл, не разделяемый никакими процессами и запись в системной таблице открытых файлов.

Далее вызовом функции `fdopen()` создаются потоки данных `fs1` и `fs2`, связанные с файлом, описанным дескриптором `fd`.

Функция `setvbuf()` меняет тип буферизации на блочную, размер блока в программе – 20 символов. Вследствие буферизации в `buff_1` будет помещена строка “Abcdefghijklmnopqrst”, а в `buff_2` – “uvwxyz”. В результате попеременного вывода символов из буферов получается строка “Aubvcwdxeyfzghijklmnopqrst”.

Созданные дескрипторы и связь между ними:



## Программа 2

Код программы:

```
#include <fcntl.h>
#include <unistd.h>

int main()
{
    char c;
    // have kernel open two connection to file alphabet.txt
    int fd1 = open("alphabet.txt", O_RDONLY);
    int fd2 = open("alphabet.txt", O_RDONLY);

    // read a char & write it alternatingly from connections fs1 & fd2
    while (read(fd1, &c, 1) == 1) {
        write(1, &c, 1);
        if (read(fd2, &c, 1) == 1)
            write(1, &c, 1);
    }

    return 0;
}
```

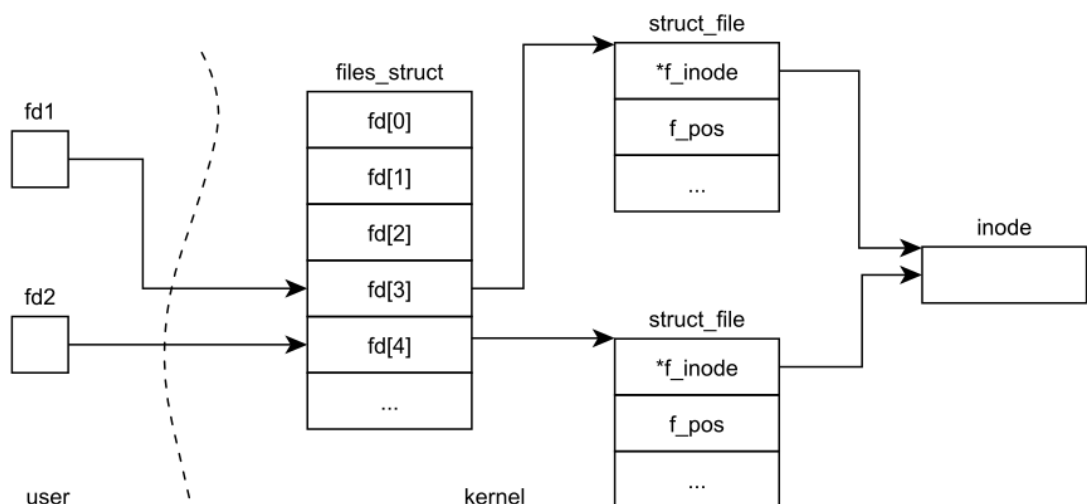
Результат работы программы:

```
oleg@Moxxx1e:~/Документы/OS/OS/lab_05$ ./prog_2.out
AAbbccddeeffgghhiijjkkllmmnnooppqrrssttuuvvwwxxyyzz
```

Анализ полученного результата:

В данной программе создаются два файловых дескриптора и две записи в таблице открытых файлов, поэтому указатели в файлах независимы друг от друга. Вследствие попеременного вывода символа то из одной строки, то из другой выводится строка, которая представлена на скриншоте.

Созданные дескрипторы и связь между ними:



### Программа 3

Код программы:

```
#include <stdio.h>

#define FILE_NAME "file.txt"

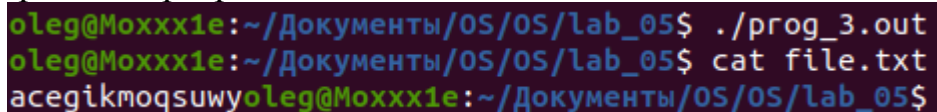
int main()
{
    FILE* fd1 = fopen(FILE_NAME, "w");
    FILE* fd2 = fopen(FILE_NAME, "w");

    for (char letter = 'a'; letter <= 'z'; letter++)
    {
        if (letter % 2 == 0)
            fprintf(fd1, "%c", letter);
        else
            fprintf(fd2, "%c", letter);
    }

    fclose(fd1);
    fclose(fd2);

    return 0;
}
```

Результат работы программы:



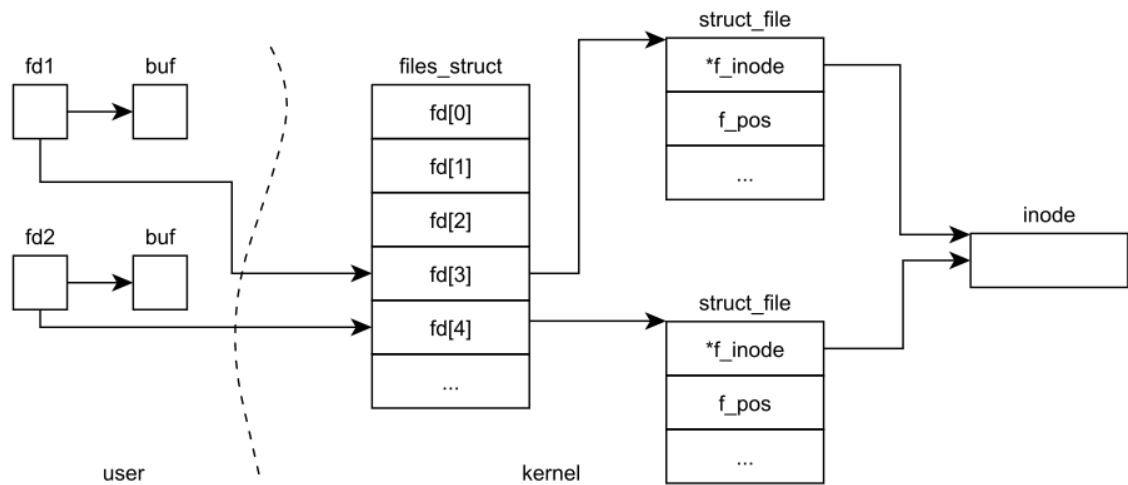
```
oleg@Moxxx1e:~/Документы/OS/OS/lab_05$ ./prog_3.out
oleg@Moxxx1e:~/Документы/OS/OS/lab_05$ cat file.txt
acegikmoqsuwyoleg@Moxxx1e:~/Документы/OS/OS/lab_05$
```

Анализ полученного результата:

Данная программа демонстрирует, что стандартный ввод-вывод буферизуется. В цикле в результате вызовов функции `fprintf()`, заполняются два буфера — в первый буфер записываются чётные символы (b, d, f...), во второй – нечётные (a, c, e...).

После вызова `fclose(fd1)` в файл запишется строка из первого буфера. После вызова `fclose(fd2)` эти данные будут удалены (файл открыт на запись) и запишется строка из второго буфера.

Созданные дескрипторы и связь между ними:



Структура FILE:

```
// «bits/types/file.h»
#ifndef __FILE defined
#define __FILE_defined 1
struct _IO_FILE;
typedef struct _IO_FILE FILE;
#endif

// «bits/libio.h»
struct _IO_FILE
{
    int _flags;           /* High-order word is _IO_MAGIC; rest is flags. */
    /* The following pointers correspond to the C++ streambuf protocol. */
    char *_IO_read_ptr;   /* Current read pointer */
    char *_IO_read_end;   /* End of get area. */
    char *_IO_read_base;  /* Start of putback+get area. */
    char *_IO_write_base; /* Start of put area. */
    char *_IO_write_ptr;  /* Current put pointer. */
    char *_IO_write_end;  /* End of put area. */
    char *_IO_buf_base;   /* Start of reserve area. */
}
```

```

char *_IO_buf_end;      /* End of reserve area. */
/* The following fields are used to support backing up and undo. */
char *_IO_save_base; /* Pointer to start of non-current get area. */
char *_IO_backup_base; /* Pointer to first valid character of backup area */
char *_IO_save_end; /* Pointer to end of non-current get area. */
struct _IO_marker *_markers;
struct _IO_FILE *_chain;
int _fileno;
int _flags2;
__off_t _old_offset; /* This used to be _offset but it's too small. */
/* 1+column number of pbase(); 0 is unknown. */
unsigned short _cur_column;
signed char _vtable_offset;
char _shortbuf[1];
_IO_lock_t *_lock;
#ifdef _IO_USE_OLD_IO_FILE
};

```

## Вывод

Функции языка С обеспечивают буферизованный ввод-вывод, а системные вызовы – небуферизованный.

В случае, если операции чтения и записи в файл выполняются последовательно, буферизация представляется действительно полезной и обеспечивающей высокую скорость выполнения. Однако существует ряд проблем, например данные могут быть не записаны в файл, а содержаться в системном буфере. Эти особенности нужно учитывать.

При работе с системными вызовами необходимо помнить, что при каждом вызове `open()` создается новый дескриптор и запись в таблице открытых файлов, у каждой записи свой указатель (`f_pos`) на позицию в файле. Это может привести к ошибкам, в ситуации, когда один и тот же файл открывается несколько раз.