



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

По курсу: «Операционные системы»

*На тему: «Буферизованный и не буферизованный
ввод-вывод»*

Студентка ИУ7-65Б
Оберган Т.М

Преподаватель
Рязанова Н.Ю.

Москва, 2020 г.

Оглавление

Структура _IO_FILE(/usr/include/bits/types/struct_FILE.h):.....	3
Первая программа	4
Листинг первой программы:	4
Результаты работы первой программы:	5
Связь структур:	5
Анализ работы первой программы:	5
Вторая программа.....	6
Листинг второй программы:.....	6
Результат работы второй программы:	7
Связь структур:	7
Анализ работы второй программы:	7
Третья программа	8
Листинг третьей программы:	8
Результат работы третьей программы:.....	8
Связь структур:	9
Анализ работы третьей программы:.....	9

Задача: анализ особенностей работы функций ввода-вывода в UNIX/Linux.

В лабораторной работе анализируется результат выполнения трех программ. Программы демонстрируют открытие одного и того же файла несколько раз. Необходимо проанализировать структуру FILE, работу программ и объяснить результаты их работы, привести рисунок, демонстрирующий созданные дескрипторы и связь между ними.

Файл /usr/include/bits/types/FILE.h:

```
typedef struct _IO_FILE FILE;
```

Структура _IO_FILE(/usr/include/bits/types/struct_FILE.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; // индекс в массиве fd структуры files_struct
    int _flags2;
    ...

    _IO_lock_t *_lock;
};
```

Первая программа

Листинг первой программы:

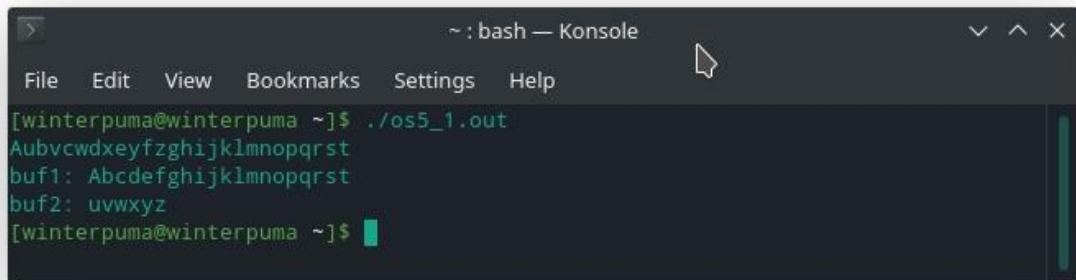
```
//testCIO.c
#include <stdio.h>
#include <fcntl.h>
#define FILENAME "alphabet.txt"

int main()
{
    // have kernel open connection to file alphabet.txt
    int fd = open(FILENAME, O_RDONLY);
    if (fd == -1)
    {
        printf("Error opening fd %s\n", FILENAME);
        return -1;
    }
    // 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);

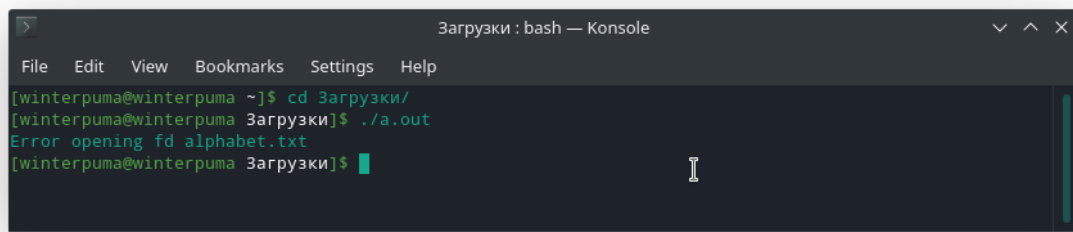
    // read a char & write it alternatingly from fs1 and fs2
    int flag1 = 1, flag2 = 2;
    while(flag1 == 1 || flag2 == 1)
    {
        char c;
        flag1 = fscanf(fs1, "%c", &c);
        if (flag1 == 1) {
            fprintf(stdout, "%c", c);
        }
        flag2 = fscanf(fs2, "%c", &c);
        if (flag2 == 1) {
            fprintf(stdout, "%c", c);
        }
    }
    fprintf(stdout, "\n");
    fprintf(stdout, "buf1: %s\nbuf2: %s\n", buff1, buff2);
    return 0;
}
```

Результаты работы первой программы:



```
[winterpuma@winterpuma ~]$ ./os5_1.out
Aubvcwdxeyfzghijklmnopqrst
buf1: Abcdefghijklmnopqrst
buf2: uvwxyz
[winterpuma@winterpuma ~]$
```

Рис.1 – результат работы первой программы



```
[winterpuma@winterpuma ~]$ cd Загрузки/
[winterpuma@winterpuma Загрузки]$ ./a.out
Error opening fd alphabet.txt
[winterpuma@winterpuma Загрузки]$
```

Рис.2 – результат работы, если файла не существует

Связь структур:

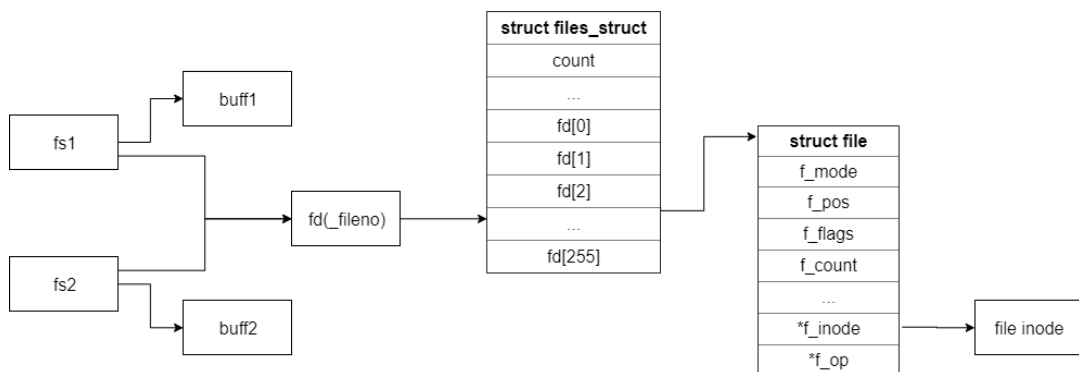


Рис.3 - связь структур первой программы

Анализ работы первой программы:

С помощью системного вызова `open()` создается дескриптор открытого на чтение файла. Системный вызов `open()` возвращает индекс в массиве `fd` структуры `files_struct`. Далее с помощью функции `fdopen()` создается `fs1` и `fs2` - структуры типа `FILE`,

связанные с fd. Создаются два буфера buff1 и buff2 размером 20 байт, с помощью функции setvbuf() для fs1 и fs2 задаются соответствующие буферы и изменяется тип буферизации на полную.

Далее в цикле происходит поочередный вызов fscanff() для fs1 и fs2. Так как установлена полная буферизация, то при вызове fscanff() буфер будет заполнен полностью либо вплоть до конца файла, а f_pos установится на следующий за последним записанным в буфер символ. Также внутри цикла будут поочередно выводиться символы из buff1 и buff2 до тех пор, пока символы в одном из буферов не закончатся. Тогда на экран будут последовательно выведены оставшиеся символы из другого буфера.

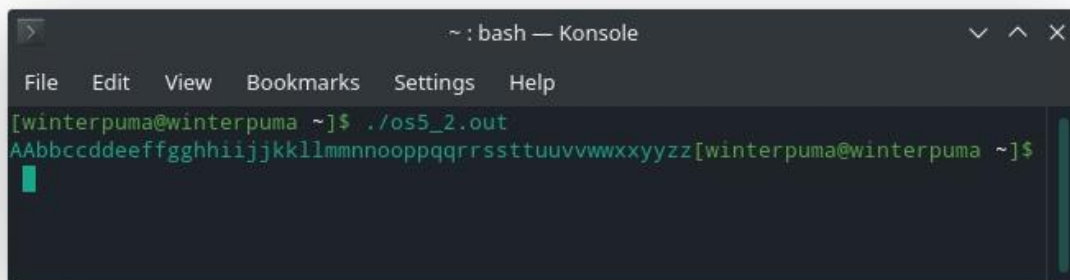
Вторая программа

Листинг второй программы:

```
#include <fcntl.h>
#include <unistd.h>
int main()
{
    char c;
    int flag = 1;

    // 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(flag)
    {
        if (read(fd1, &c, 1) == 1)
        {
            write(1, &c, 1);
            if (read(fd2, &c, 1) == 1)
                write(1, &c, 1);
            else
                flag = 0;
        }
        else
        {
            flag = 0;
        }
    }
    return 0;
}
```

Результат работы второй программы:



```
~ : bash — Konsole
File Edit View Bookmarks Settings Help
[winterpuma@winterpuma ~]$ ./os5_2.out
AAbbccddeeffgghhiijjkkllmmnnoppqrrssttuuvvwxxyyzz[winterpuma@winterpuma ~]$
```

Рис. 4 - результат работы второй программы

Связь структур:

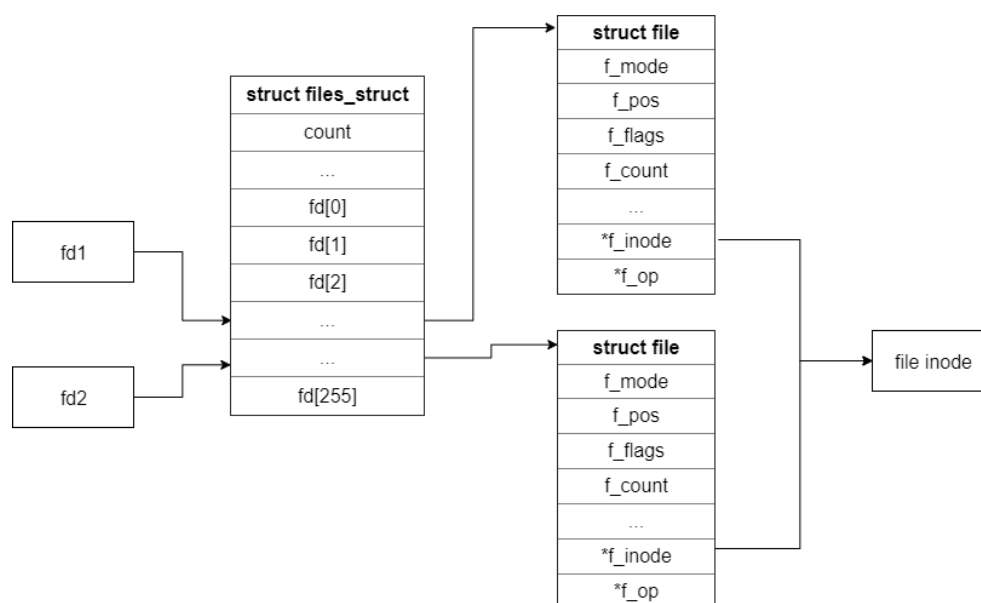


Рис.5 - связь структур второй программы

Анализ работы второй программы:

С помощью системного вызова `open()` создастся два дескриптора открытого на чтение файла. Так как дескриптора два, то чтение будет проходить независимо (разные `f_pos`). На экран дважды напечатаются все символы из `alphabet.txt`.

Третья программа

Листинг третьей программы:

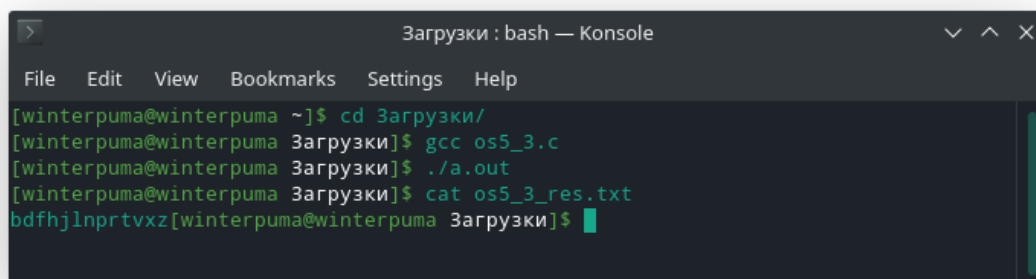
```
#include <stdio.h>
#define FILENAME "os5_3_res.txt"

int main()
{
    FILE *fs1 = fopen(FILENAME, "w");
    FILE *fs2 = fopen(FILENAME, "w");

    for (char c = 'a'; c <= 'z'; c++)
    {
        if (c % 2)
            fprintf(fs1, "%c", c);
        else
            fprintf(fs2, "%c", c);
    }

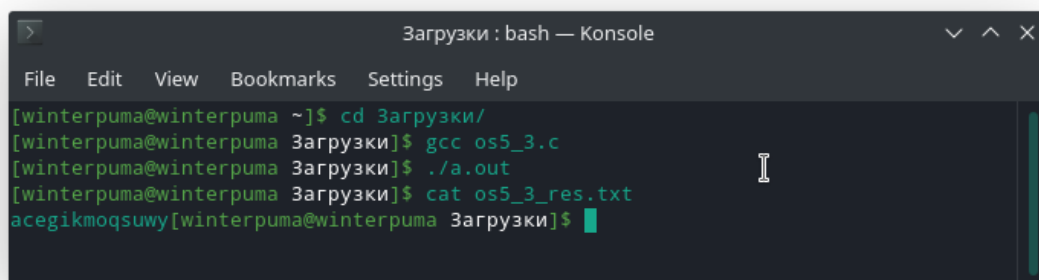
    fclose(fs1);
    fclose(fs2);
    return 0;
}
```

Результат работы третьей программы:



```
Загрузки : bash — Konsole
File Edit View Bookmarks Settings Help
[winterpuma@winterpuma ~]$ cd Загрузки/
[winterpuma@winterpuma Загрузки]$ gcc os5_3.c
[winterpuma@winterpuma Загрузки]$ ./a.out
[winterpuma@winterpuma Загрузки]$ cat os5_3_res.txt
bdfhjlnprtvxz[winterpuma@winterpuma Загрузки]$
```

Рис. 6 - результат работы третьей программы



```
Загрузки : bash — Konsole
File Edit View Bookmarks Settings Help
[winterpuma@winterpuma ~]$ cd Загрузки/
[winterpuma@winterpuma Загрузки]$ gcc os5_3.c
[winterpuma@winterpuma Загрузки]$ ./a.out
[winterpuma@winterpuma Загрузки]$ cat os5_3_res.txt
acegikmoqsuwy[winterpuma@winterpuma Загрузки]$
```

Рис.7 - результат работы третьей программы, если поменять порядок fclose()

Связь структур:

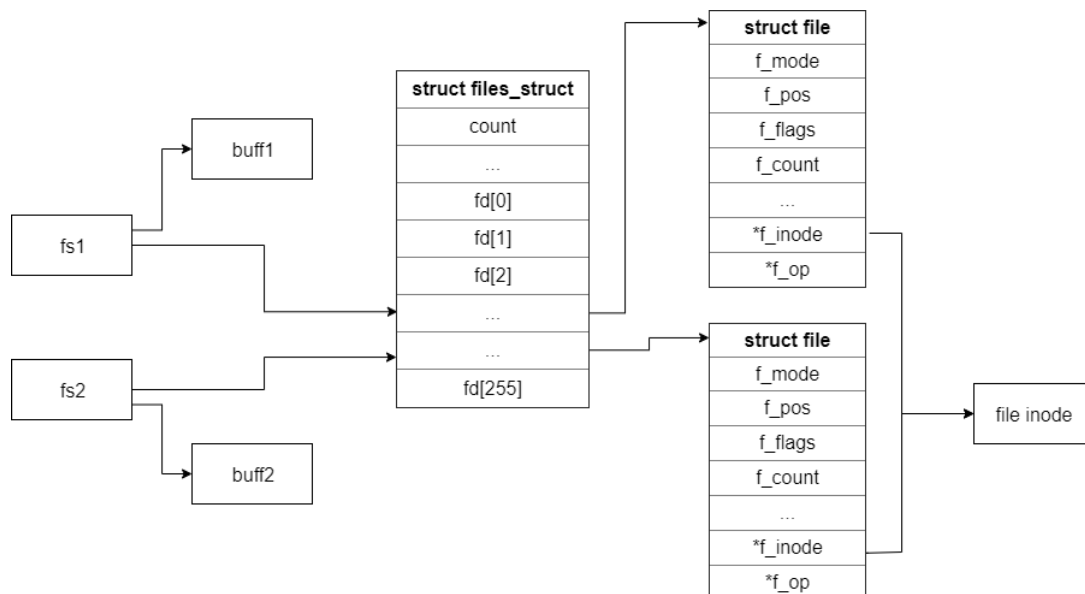


Рис. 8 - связь структур третьей программы

Анализ работы третьей программы:

Программа открывает один и тот же файл на запись два раза с использованием библиотечной функции `open()`. Для этого объявляются два файловых дескриптора. По умолчанию используется полная буферизация, при которой запись в файл из буфера произойдет либо при заполнении буфера, либо при вызове `fclose()`, либо при завершении процесса.

В цикле записываются в файл буквы латинского алфавита поочередно передавая функции `fprintf()` то первый дескриптор, то – второй.

При вызове `fclose()` для `fs1` `buff1` записывается в файл. При вызове `fclose()` для `fs2`, все содержимое файла очищается, а в файл записывается содержимое `buff2`.

В итоге произошла утеря данных, в файле окажется только содержимое `buff2`.