**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 ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

## По курсу: "Операционные системы"

Тема Буферизованный и не буферизованный ввод-вывод

Группа ИУ7-63Б

Студент Сукочева А.

Преподаватель Рязанова Н.Ю.

Москва — 2021 г.

# Практическая часть

**Задание 1.** Первая программа.

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

#define OK 0
#define FILE_NAME "alphabet.txt"
#define BUFF_SIZE 20

#define GREEN "\33[32m"
#define BLUE "\33[34m"
#define RED "\33[31m"

int main()
{
    int fd = open(FILE_NAME, O_RDONLY);

    FILE *fs1 = fdopen(fd, "r");
    char buff1[BUFF_SIZE];
    setvbuf(fs1, buff1, _IOFBF, BUFF_SIZE);

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

    int flag1 = 1, flag2 = 2;

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

    return OK;
}
```

## Анализ

Данная программа считывает информацию из файла 'alphabet.txt', который содержит строку символов 'Abcdefghijklmnopqrstuvwxyz'. И при помощи двух буферов посимвольно выводит считанные символы в стандартный поток вывода stdout.

Зеленым цветом показан вывод при помощи первого буфера, синим при помощи второго буфера.

В файле '/usr/include/x86_64-linux-gnu/bits/types/FILE.h' было создано дополнительное имя для структуры FILE, которая используется в нашей программе.

```
        typedef struct _IO_FILE FILE;
```

В файле 'cat /usr/include/x86_64-linux-gnu/bits/libio.h' находится описание структуры _IO_FILE

```
struct _IO_FILE
{
    int _flags; /* High-order word is _IO_MAGIC; rest is flags. */
#define _IO_file_flags _flags

    /* The following pointers correspond to the C++ streambuf protocol. */
    /* Note:  Tk uses the _IO_read_ptr and _IO_read_end fields directly. */
    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;
#if 0
    int _blksize;
#else
    int _flags2;
#endif
    _IO_off_t _old_offset; /* This used to be _offset but it's too small.  */

#define __HAVE_COLUMN /* temporary */
    /* 1+column number of pbase(); 0 is unknown. */
    unsigned short _cur_column;
    signed char _vtable_offset;
    char _shortbuf[1];

    /*  char* _save_gptr;  char* _save_egptr; */

    _IO_lock_t *_lock;
#ifdef _IO_USE_OLD_IO_FILE
};
```

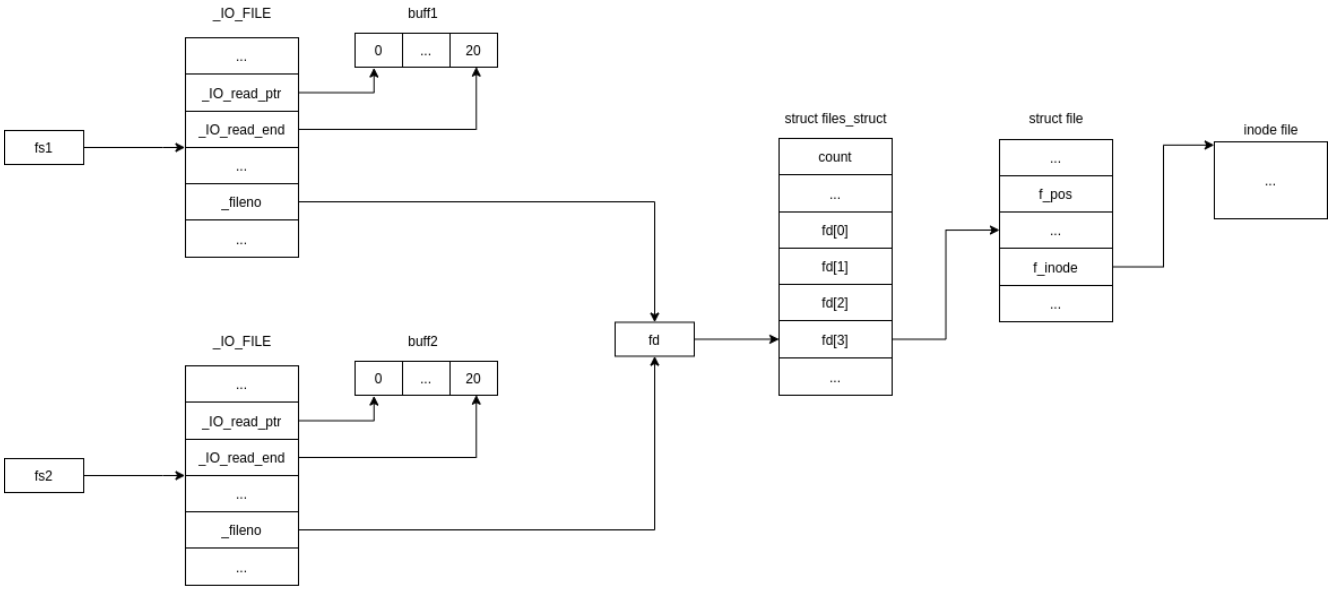


Рис. 1: Связь между дескрипторами в первой программе

В начале функции main open() создает новый файловый дескриптор для открытого только на чтение (O_RDONLY) файла 'alphabet.txt', запись в системной таблице открытых файлов. Эта запись регистрирует смещение в файле и флаги состояния файла.

Далее fdopen() создает два указателя на структуру FILE, приведенную выше. В данных структурах поле _fileno будет содержать дескриптор, который вернула функция fopen(). Для fs1 и fs2 эти поля будут равны 3.

Функция setvbuf() изменяет тип буферизации для fs1 и fs2 на полную буферизацию, а также явно задает размер буфера 20 байт.

Далее при первом вызове fscanf() буфер fs1 заполнится полностью, т.е. первыми 20 символами. Значение f_pos в структуре struct_file открытого файла увеличится на 20. Далее при первом вызове fscanf() для fs2 в buff2 считаются оставшиеся 6 символов, начиная с f_pos (т.к. fs1 и fs2 ссылаются на один и тот же дескриптор fd).

Далее в цикле поочередно выводятся символы из buff1 и buff2. Т.к. в buff2 записались оставшиеся 6 символов, после 6 итерации цикла будут выводится символы только из buff1.


Рис. 2: Результат работы первой программы

**Задание 2.** Вторая программа. Один поток.

```c
        //testKernelIO.c
#include <fcntl.h>
#include <unistd.h> // read, write.

#define OK 0
#define FILE_NAME "alphabet.txt"

int main()
{
        char c;

        int fd1 = open(FILE_NAME, O_RDONLY);
        int fd2 = open(FILE_NAME, O_RDONLY);

        while (read(fd1, &c, 1) && read(fd2, &c, 1))
        {
                write(1, &c, 1);
                write(1, &c, 1);
        }

        write(1, "\n", 1);
        return OK;
}
```

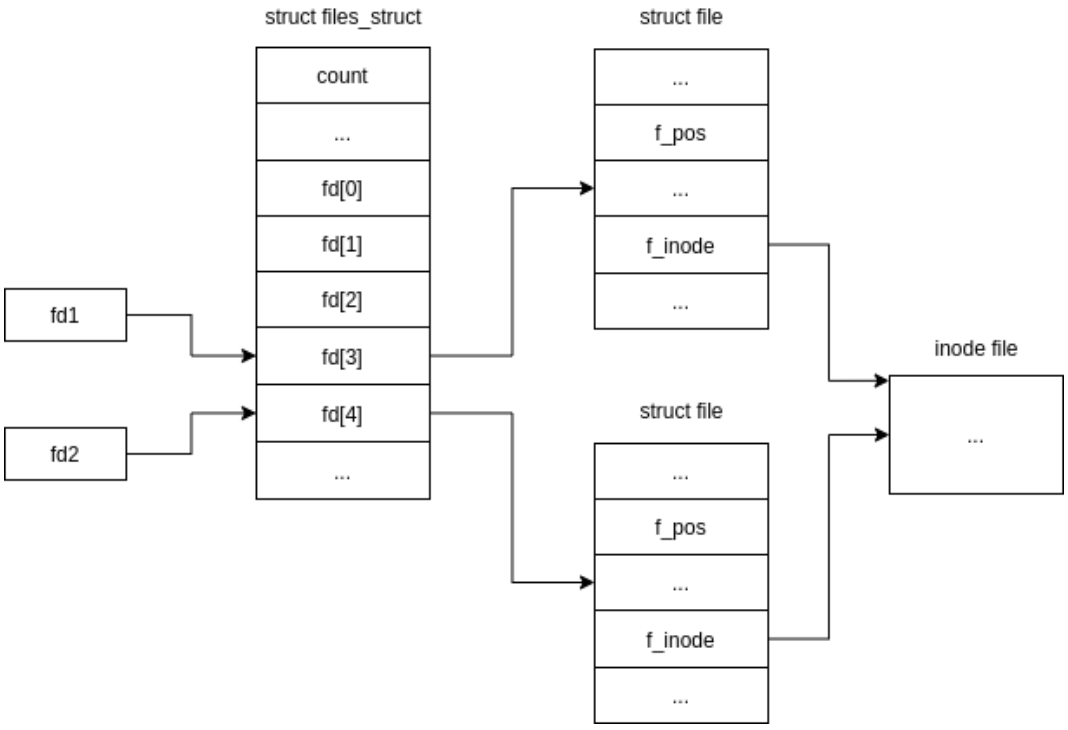


Рис. 3: Связь между дескрипторами во второй программе

### Анализ

В данной программе создается два файловых дескриптора при помощи функции open(). При этом создается две разные структуры struct_file, описывающие файл. Далее в цикле поочередно считываются символы из файла и выводятся на экран. Т.к. созданы две структуры struct file, то у каждой структуры будет свой f_pos и смещения в файловых дескрипторах будут независимы, поэтому на экран будут дважды выводится символы одного и того же файла.

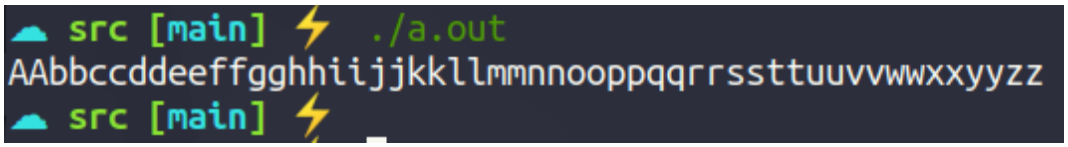


Рис. 4: Результат работы второй программы

Вторая программа. Два потока.

```c
#include <fcntl.h>
#include <unistd.h> // read, write.
#include <pthread.h>
#include <stdio.h>

#define OK 0
#define ERROR_THREAD_CREATE -1
#define FILE_NAME "alphabet.txt"

void read_file(int fd)
{
    char c;
    while (read(fd, &c, 1))
    {
        write(1, &c, 1);
    }
}

void *thr_fn(void *arg)
{
    int fd = open(FILE_NAME, O_RDONLY);
    read_file(fd);
}

int main()
{
    pthread_t tid;

    int fd = open(FILE_NAME, O_RDONLY);

    int err = pthread_create(&tid, NULL, thr_fn, 0);
    if (err)
    {
        return ERROR_THREAD_CREATE;
    }

    read_file(fd);
    pthread_join(tid, NULL);

    return OK;
}
```

В программе также, как и при реализации с одним потоком, создается два файловых дескриптора для открытого файла, записи в системной таблице открытых файлов. У каждой записи будет свое смещение f_pos. Т.к. главный поток ждет окончания дочернего, то гарантируется вывод всего алфавита дважды. Порядок, в котором будут выводится символы алфавита, неизвестен, т.к. вывод производится параллельно.

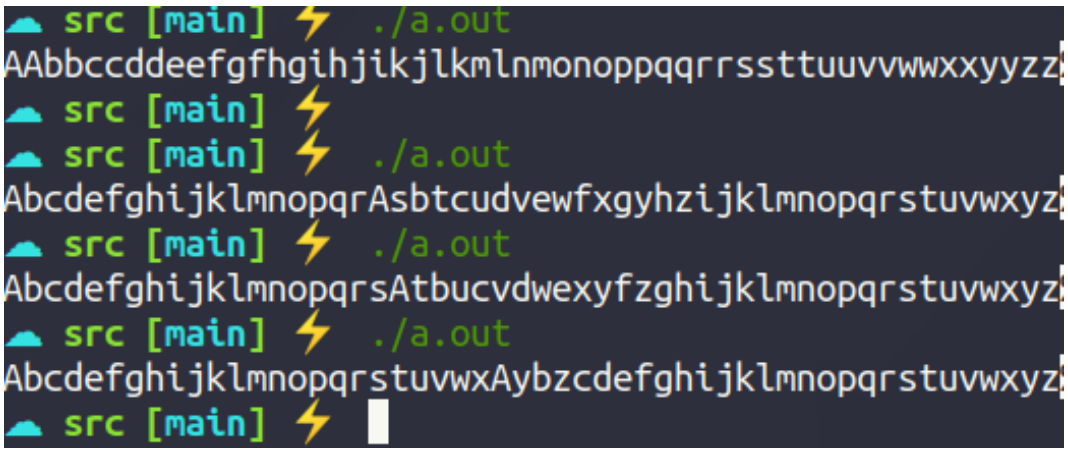


Рис. 5: Результат работы второй программы при двух потоках

**Задание 3.** Третья программа. Один поток.

```c
#include <stdio.h>
#include <sys/stat.h>

#define FILE_NAME "task3.txt"
#define OK 0

void Info()
{
    struct stat statbuf;

    stat(FILE_NAME, &statbuf);
    printf("inode: %ld\n", statbuf.st_ino);
    printf("st_size: %ld\n", statbuf.st_size);
    printf("st_blksize: %ld\n\n", statbuf.st_blksize);
}

int main()
{

    FILE *f1 = fopen(FILE_NAME, "w");
    Info();

    FILE *f2 = fopen(FILE_NAME, "w");
    Info();

    char c = 'a'; // 97
    while (c <= 'z')
    {
        if (c % 2)
        {
            fprintf(f1, "%c", c); // acej...
        }
        else
        {
            fprintf(f2, "%c", c); // bdfh...
        }
        c++;
```

```
    }

    fclose(f1);
    Info();

    fclose(f2);
    Info();

    return OK;
}
```

В данной программе файл 'task3.txt' открывается 2 раза для записи. Выполняется ввод через стандартную библиотеку C (stdio.h). fprintf() - буферизованный ввод/вывод. Буфер создается без нашего явного вмешательства. Сначала информация пишется в буфер, а из буфера информация переписывается в файл в результате 3-ех действий:

1. буфер полон;
2. принудительная запись fflush();
3. если вызван fclose().

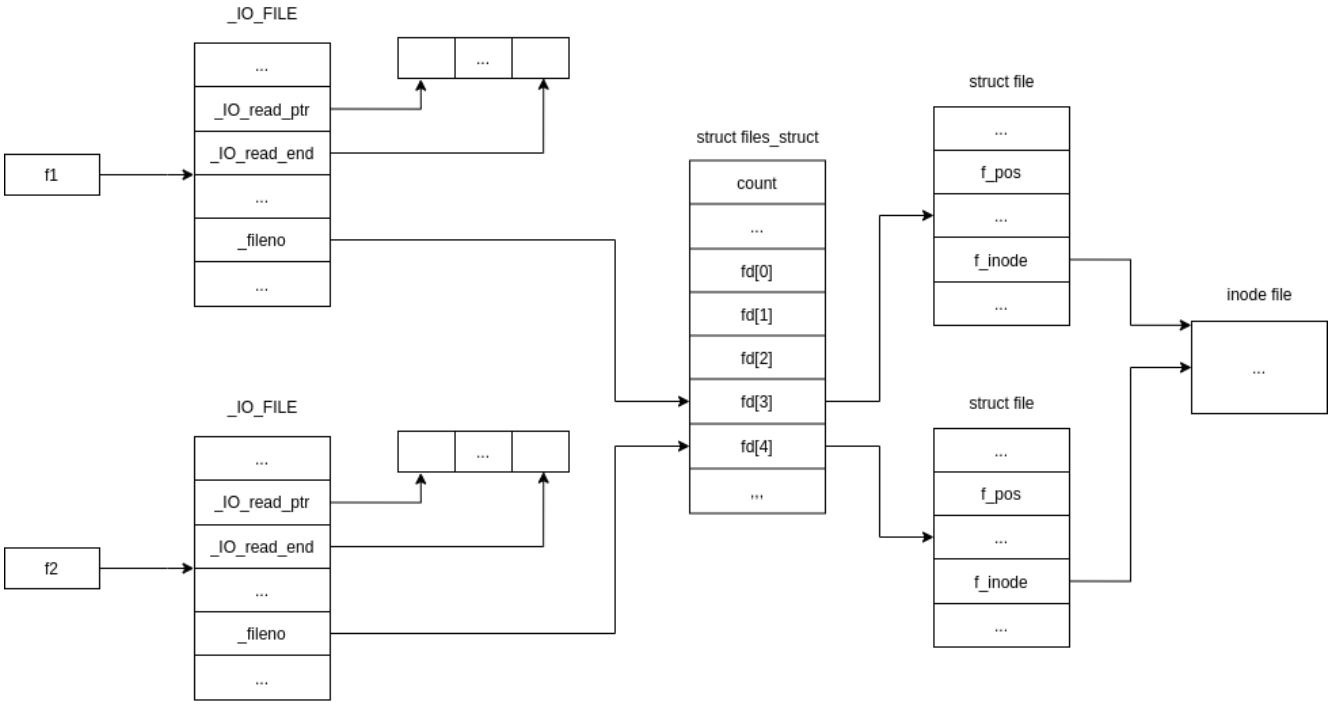


Рис. 6: Связь между дескрипторами в третьей программе

В нашей программе символы, имеющие нечетный код в таблице ASCII записываются в буфер, который находится в дескрипторе f1, в f2 соответственно записываются четные. Таким образом в буфере, который содержится в f1 будут символы: 'acej...', а в f2 'bdfh...'. В нашем случае информация из фубера запишется в файл при вызове fclose(). Т.к. f_pos независимы у каждого дескриптора файла, то при закрытии файла запись будет производиться начиная с начала файла в обоих случаях. Таким образом информация, которая будет записана в файл, после первого вызова fclose() будет потеряна в результате второго вызова fclose() рис. 7.

Если поменять вызовы fclose() местами, то будет потеряна информация, которая содержится во втором буфере рис. 8.

```
    fclose(f2);
    fclose(f1);
```

```
src [main] cat task3.txt
bdfhjlnprtvxz%
```

Рис. 7: Результат работы третьей программы

```
src [main] cat task3.txt
acegikmoqsuwy%
```

Рис. 8: Результат работы третьей программы с другим порядком вызовов fclose()

С помощью stat после каждого вызова fopen() и fclose() показана некоторая информация о файле рис. 10.

```
src [main] ./a.out
inode: 32309811
Общий размер в байтах: 0
Размер блока ввода-вывода: 4096

inode: 32309811
Общий размер в байтах: 0
Размер блока ввода-вывода: 4096

inode: 32309811
Общий размер в байтах: 13
Размер блока ввода-вывода: 4096

inode: 32309811
Общий размер в байтах: 13
Размер блока ввода-вывода: 4096
```

Рис. 9: Информация, полученная при помощи stat

```
src [main] ls -i | grep task3.txt
32309811 task3.txt
src [main]
```

Рис. 10: inode файла task3.txt

Третья программа. Два потока.

```
#include <stdio.h>
#include <sys/stat.h>
#include <pthread.h>

#define FILE_NAME "task3_pthread.txt"
#define ERROR_THREAD_CREATE -1
#define OK 0

void Info()
{
    struct stat statbuf;

    stat(FILE_NAME, &statbuf);
    printf("inode: %ld\n", statbuf.st_ino);
    printf("st_size: %ld\n", statbuf.st_size);
    printf("st_blksize: %ld\n\n", statbuf.st_blksize);
}

void WriteToFile(char c)
{
    FILE *f = fopen(FILE_NAME, "w");
    Info();

    while (c <= 'z')
    {
        fprintf(f, "%c", c);
        c += 2;
    }

    fclose(f);
    Info();
}

void *thr_fn(void *arg)
{
    WriteToFile('b');
}

int main()
{
    pthread_t tid;

    int err = pthread_create(&tid, NULL, thr_fn, NULL);
    if (err)
    {
        return ERROR_THREAD_CREATE;
    }

    WriteToFile('a');
    pthread_join(tid, NULL);

    return OK;
}
```

В данной программе создается поток. Главный поток записывает в файл символы, начиная с 'a', в то время, как созданный нами поток записывает символы, начиная с 'b'. Так же как и в приведенной выше программе с одним потоком происходит потеря данных. Данные будут записаны из того буфера (который содержится в дескрипторе), для которого будет вызван fclose() последним, потому что он перезапишет данные с начала файла. Можно принудительно в главном потоке вызвать sleep(), чтобы в файле были данные записанные из главного потока.

Тогда результат работы представлен на рис. 8.

```
        sleep(1);
        WriteToFile('a');
```

# Заключение

В ходе выполнения данной лабораторной работы были проанализированы три программы. Открытие файла при помощи open() создает новый файловый дескриптор для открытого файла, запись в системной таблице открытых файлов. У различных дескрипторов открытого файла смещения не зависят друг от друга. Поэтому чтобы избежать потери данных необходимо учитывать, что файл может быть открыт несколько раз.