



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           ИУ7-62Б

Студент         Козлова И.В.

Преподаватель Рязанова Н.Ю.

## Практическая часть

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;
#ifdef 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
};
```

### Задание 1. Первая программа один поток.

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

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

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

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

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

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

    printf("\nfs1 _fileno: %d\n", fs1->_fileno);
    printf("\nfs2 _fileno: %d\n", fs2->_fileno);
    printf("\nfs1 buff1[0] == fs1->_IO_buf_base: %d\n", buff1 ==
        fs1->_IO_buf_base);
    printf("\nfs2 buff2[0] == fs2->_IO_buf_base: %d\n", buff2 ==
        fs2->_IO_buf_base);

    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);
        }
    }

    printf("\n");
    return 0;
}
```

### Объяснение

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.

Так как отчет печатался на Ч/Б принтере, различия в цветах не видны.

Зеленым цветом показан вывод с помощью первого буфера, синим с помощью второго.

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.

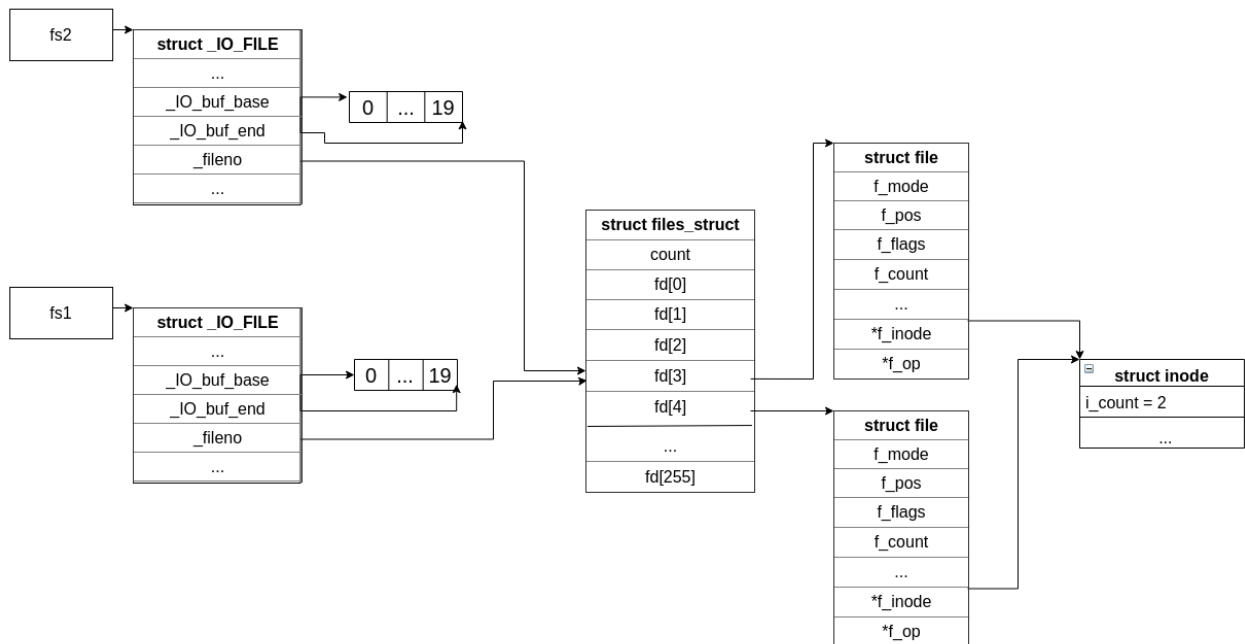


Рис. 1: Связь между дескрипторами в первой программе

Далее `fdopen()` создает два указателя на структуру `FILE`, приведенную выше. В данных структурах поле `_fileno` будет содержать дескриптор, который вернула функция `fdopen()`. 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` считываются 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 выводиться символы только из `buff1`.

```

kpirap18@kpirap18-Aspire-A514-54:~/sem6/BMSTU-OS-sem6/lab_05$ make app1
gcc proc_01.c
kpirap18@kpirap18-Aspire-A514-54:~/sem6/BMSTU-OS-sem6/lab_05$ ./a.out
AUBVCWDXEYFZGHIJKLMNOPQRST
  
```

Рис. 2: Результат работы первой программы

Первая программа два потока.

```

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

#define BUF_SIZE 20

void *run_buffer(void *args)
{
    int flag = 1;
    FILE *fs = (FILE *)args;

    while (flag == 1){
        char c;
        if ((flag = fscanf(fs, "%c\n", &c)) == VALID_READED) {
            fprintf(stdout, "thread_2: " "%c\n", c);
        }
    }
}
  
```

```

        return NULL;
    }

    int main(void)
    {
        setbuf(stdout, NULL);

        pthread_t thread;
        int fd = open("alphabet.txt", O_RDONLY);

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

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

        int rc = pthread_create(&thread, NULL, run_buffer, (void *)fs2);

        int flag = 1;
        while (flag == 1){
            char c;
            flag = fscanf(fs1, "%c\n", &c);
            if (flag == 1){
                fprintf(stdout, "thread_1: " "%c\n", c);
            }
        }

        pthread_join(thread, NULL);

        return 0;
    }

```

```

kpirap18@kpirap18-Aspire-A514-54:~/sem6/BMSTU-05-sem6/lab_05$ ./a.out
thread 1: A
thread 1: B
thread 1: C
thread 1: D
thread 1: E
thread 1: F
thread 1: G
thread 1: H
thread 1: I
thread 1: J
thread 1: K
thread 1: L
thread 1: M
thread 1: N
thread 1: O
thread 1: P
thread 1: Q
thread 1: R
thread 1: S
thread 1: T
thread 2: U
thread 2: V
thread 2: W
thread 2: X
thread 2: Y
thread 2: Z

```

Рис. 3: Результат работы первой программы

## Задание 2. Вторая программа. Один поток.

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

int main()
{
    char c;

    int fd1 = open("alphabet.txt", O_RDONLY);
    int fd2 = open("alphabet.txt", O_RDONLY);
    int rc1, rc2 = 1;

    while (rc1 == 1 || rc2 == 1){
        char c;

        rc1 = read(fd1, &c, 1);
        if (rc1 == 1){
            write(1, &c, 1);}

        rc2 = read(fd2, &c, 1);
        if (rc2 == 1){
            write(1, &c, 1);}

    }

    write(1, "\n", 1);
    return 0;
}
```

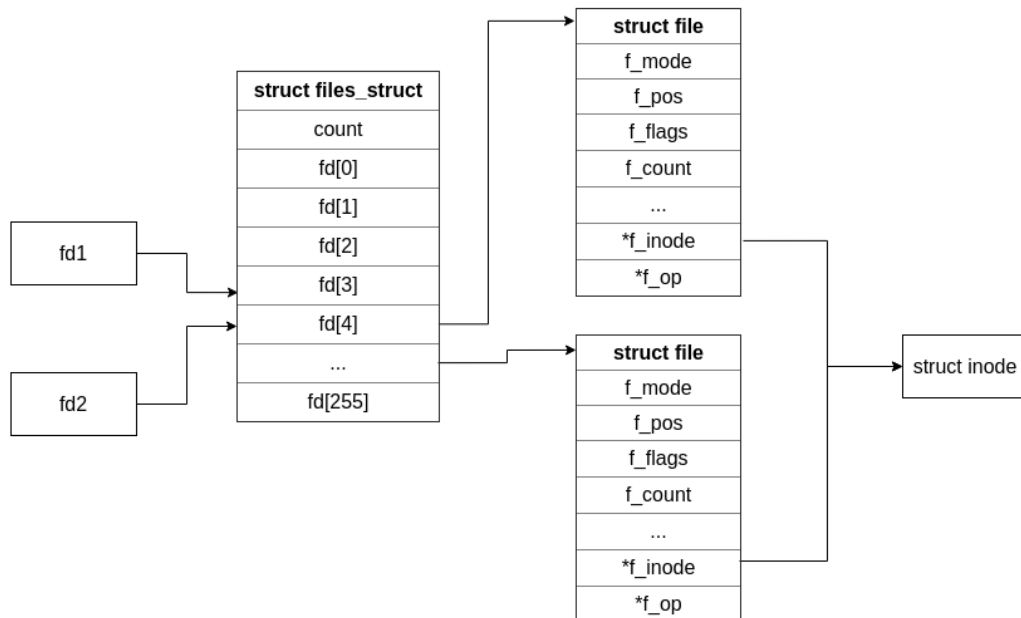
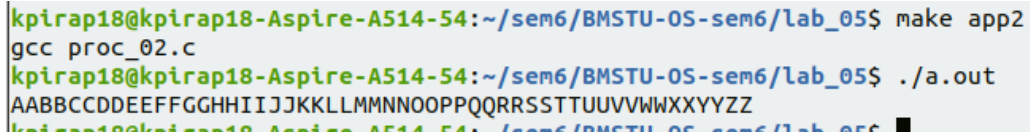


Рис. 4: Связь между дескрипторами во второй программе

## Объяснение

В данной программе создается два дескриптора открытого файла при помощи функции `open()`. В системной таблице открытых файлов создаются две новых записи. 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 выводиться символы одного и того же файла.



```
kpirap18@kpirap18-Aspire-A514-54:~/sem6/BMSTU-05-sem6/lab_05$ make app2
gcc proc_02.c
kpirap18@kpirap18-Aspire-A514-54:~/sem6/BMSTU-05-sem6/lab_05$ ./a.out
AABBCCDDEEFFGGHHIIJJKKLLMMNNOO PPPQQRRSSTTUUVVWWXXYYZZ
```

Рис. 5: Результат работы второй программы

Вторая программа. Два потока.

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

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

void *thr_fn(void *arg){
    int fd = open("alphabet.txt", O_RDONLY);
    read_file(fd);
}

int main()
{
    pthread_t tid;

    int fd = open("alphabet.txt", O_RDONLY);

    int err = pthread_create(&tid, NULL, thr_fn, 0);
    if (err) {
        printf("Error while create pthread");
        return -1;
    }

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

    write(1, "\n", 1);
    return 0;
}
```

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 выводиться символы алфавита, неизвестен, т.к. вывод производится параллельно.





буфера информация переписывается в файл в результате 3-х действий:

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

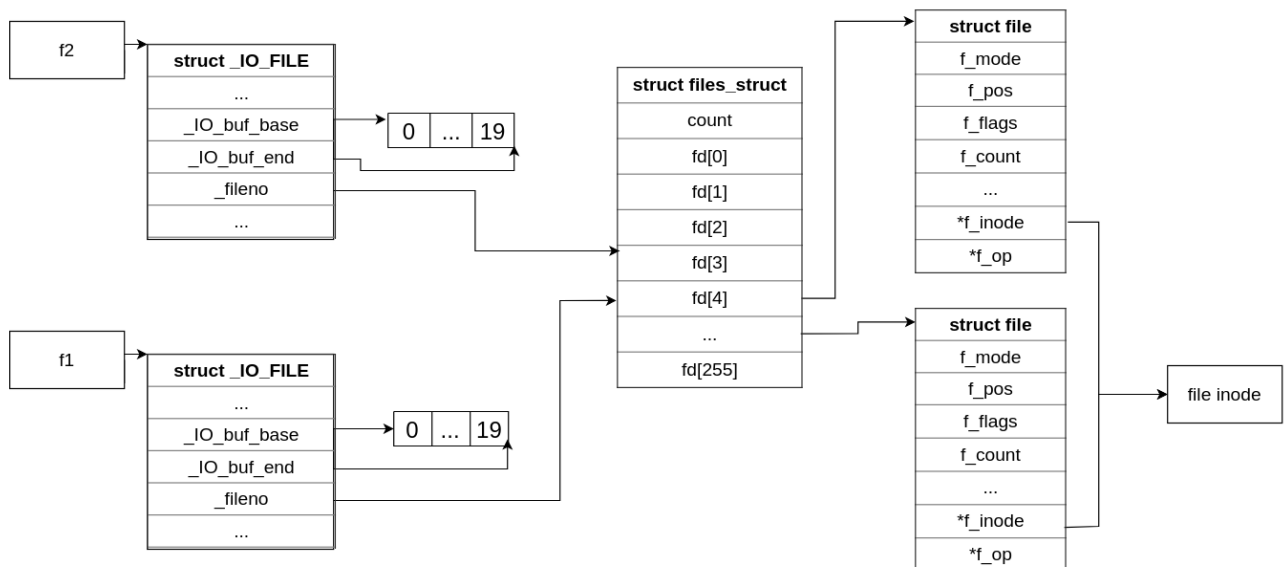


Рис. 7: Связь между дескрипторами в третьей программе

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. 8.

```
kpirap18@kpirap18-Aspire-A514-54:~/sem6/BMSTU-05-sem6/lab_05$ cat res3.txt
bdfhjlnprtvxzkpirap18@kpirap18-Aspire-A514-54:~/sem6/BMSTU-05-sem6/lab_05$
```

Рис. 8: Результат работы третьей программы

Если поменять вызовы `fclose()` местами, то будет потеряна информация, которая содержится во втором буфере рис. 9.

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

```
kpirap18@kpirap18-Aspire-A514-54:~/sem6/BMSTU-05-sem6/lab_05$ cat res3.txt
acegikmoqsuwykpirap18@kpirap18-Aspire-A514-54:~/sem6/BMSTU-05-sem6/lab_05$
```

Рис. 9: Результат работы третьей программы с другим порядком вызовов `fclose()`

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

```
kpirap18@kpirap18-Aspire-A514-54:~/sem6/BMSTU-05-sem6/lab_05$ ./a.out
inode: 7080893
Общий размер в байтах: 0
Размер блока ввода-вывода: 4096

inode: 7080893
Общий размер в байтах: 0
Размер блока ввода-вывода: 4096

fs1 _fileno: 3
fs2 _fileno: 4

inode: 7080893
Общий размер в байтах: 13
Размер блока ввода-вывода: 4096

inode: 7080893
Общий размер в байтах: 13
Размер блока ввода-вывода: 4096

kpirap18@kpirap18-Aspire-A514-54:~/sem6/BMSTU-05-sem6/lab_05$ ls -li | grep res3.txt
7080893 res3.txt
```

Рис. 10: Информация, полученная при помощи `stat`

Третья программа. Два потока.

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

void info()
{
    struct stat statbuf;

    stat("res3.txt", &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 run_buffer(char c)
{
    FILE *f = fopen("res3_th.txt", "w");
    info();

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

    fclose(f);
    info();
}

void *for_help(void *arg)
{
    run_buffer('a');
}

int main()
{
    pthread_t thread;
    int rc = pthread_create(&thread, NULL, for_help, NULL);

    if (rc){
        printf("Error while create pthread");
        return -1;
    }

    //sleep(1);
    run_buffer('b');

    pthread_join(thread, NULL);
    return 0;
}
```

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. 9.