



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

Дисциплина: Операционные системы

Тема: Буферизованный и не буферизованный ввод-вывод

Студент: Платонова О. С.

Группа: ИУ7-65Б

Оценка(баллы) _____

Преподаватель: Рязанова Н. Ю.

Москва, 2021 г.

Программа 1

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

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

    FILE *fs1 = fdopen(fd, "r");
    char buff1[20];
    setvbuf(fs1, buff1, _IOFBF, 20);

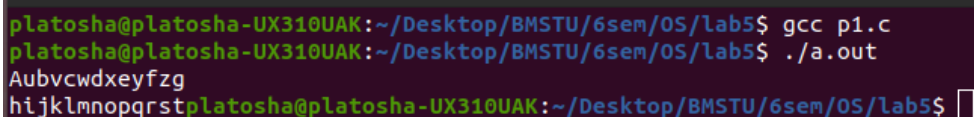
    FILE *fs2 = fdopen(fd, "r");
    char buff2[20];
    setvbuf(fs2, buff2, _IOFBF, 20);

    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;
}
```



```
platosh@platosh-UX310UAK:~/Desktop/BMSTU/6sem/OS/lab5$ gcc p1.c
platosh@platosh-UX310UAK:~/Desktop/BMSTU/6sem/OS/lab5$ ./a.out
Aubvcwdxeyfzg
hijklmnopqrstplatosh@platosh-UX310UAK:~/Desktop/BMSTU/6sem/OS/lab5$
```

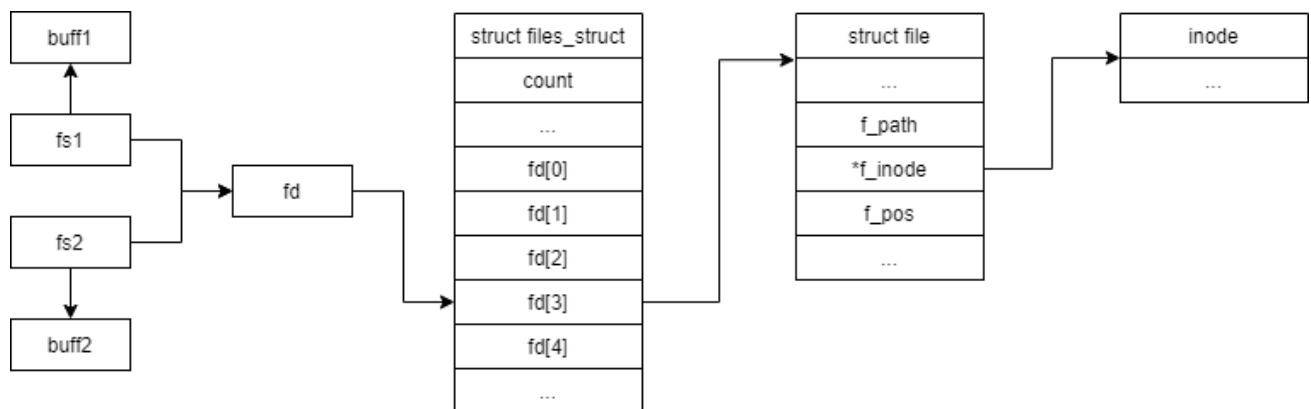
Системный вызов `open()` создаст дескриптор открытого файла. Флаг `O_RDONLY` указывает на открытие файла только на чтение. В результате вызова `open()` будут вызваны функции ядра, в том числе `do_filp_open()`. Результатом является создание дескриптора открытого файла в системной таблице открытых файлов.

Системный вызов `fdopen()`, в который передается дескриптор `fd`, создаст два объекта типа `FILE`, ссылающиеся на один дескриптор открытого файла.

Система создаст буфер на 20 байт. Вызов `setvbuf` свяжет поток, ссылающийся на открытый файл, с созданным буфером. Параметр `_IOFBF` указывает на режим полной буферизации.

В цикле будут выполнены поочередные вызовы `fscanf()`. Первый вызов заполнит буфер `buff1` первыми 20-ю символами. Указатель `f_pos` будет установлен за 20-м символом. Вызов `fprintf()` выведет на экран первый символ из `buff1` (“А”). Второй вызов `fscanf()` заполнит буфер `buff2` оставшимися символами, начиная с 21-го, поскольку оба объекта ссылаются на один дескриптор (т.е. одно значение `f_pos`). Затем будет выполнен вывод первого символа из `buff2` (“u”).

В результате поочередного вывода символов из `buff1` и `buff2`, результатом является строка “Aubvcwdxeyfzghijklmnopqrst”.



Программа 2

```

#include <fcntl.h>

int main()
{
    char c;

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

    while (read(fd1, &c, 1) == 1 &&
           read(fd2, &c, 1) == 1) {

        write(1, &c, 1);
        write(1, &c, 1);
    }

    return 0;
}

```

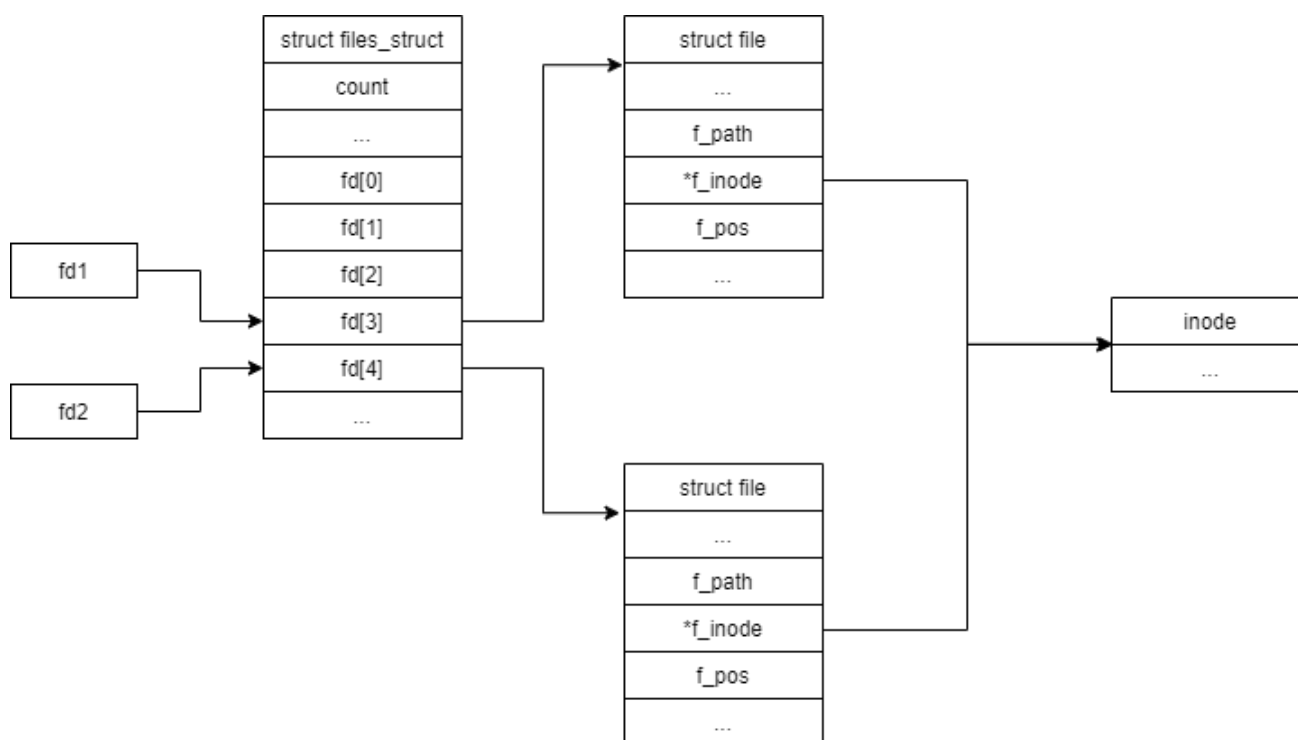
```

platosha@platosha-UX310UAK:~/Desktop/BMSTU/6sem/OS/lab5$ gcc p21.c -w
platosha@platosha-UX310UAK:~/Desktop/BMSTU/6sem/OS/lab5$ ./a.out
AAbbccddeeffgghhiijjkkllmmnnnooppqrrssttuuvvwwxxyyzz
platosha@platosha-UX310UAK:~/Desktop/BMSTU/6sem/OS/lab5$ 

```

Вызов функции `open()` создаст два дескриптора одного файла, доступного только для чтения. Т.е. будут созданы две структуры `struct file` в общесистемной таблице открытых файлов. Следовательно, поля `f_pos` в каждой из структур независимы.

В цикле выполняются `read()`, считывающий символ и `write()`, записывающий символ в стандартный поток. Указатель `f_pos` изменяется независимо от другого дескриптора, поэтому каждый символ будет выведен дважды.



Программа 2 (с потоками)

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

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

    sleep(rand() % 2);
}

int main()
{
    int fd1 = open("alphabet.txt", O_RDONLY);
    int fd2 = open("alphabet.txt", O_RDONLY);

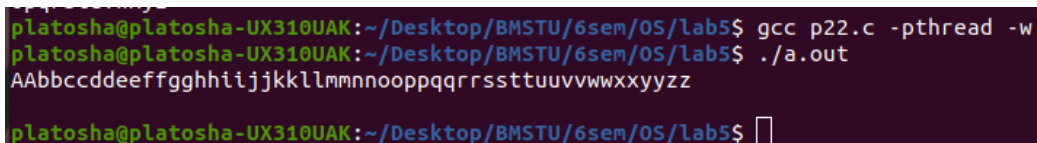
    pthread_t thread1, thread2;

    int stat1 = pthread_create(&thread1, NULL, thread_read, &fd1);
    if (stat1 != 0) {
        printf("Error. Can't create thread 1!\n");
        exit(1);
    }

    int stat2 = pthread_create(&thread2, NULL, thread_read, &fd2);
    if (stat2 != 0) {
        printf("Error. Can't create thread 2!\n");
        exit(1);
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```



```
platosh@platosh-UX310UAK:~/Desktop/BMSTU/6sem/OS/lab5$ gcc p22.c -pthread -w
platosh@platosh-UX310UAK:~/Desktop/BMSTU/6sem/OS/lab5$ ./a.out
AAbbccddeeffgghhiijjkkllmmnnooppqrrssttuuvvwxxyyzz
platosh@platosh-UX310UAK:~/Desktop/BMSTU/6sem/OS/lab5$
```

Аналогично программе без потока, вызов `open()` создаст два независимых дескриптора открытых файлов, содержащих собственные значения `f_pos`. В результате каждый поток выполнит чтение и запись независимо, что приведет к повтору каждого символа в результирующей строке.

Программа 3

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

#define wrAlphabet "wrAlphabet.txt"

int main()
{
    struct stat sb;

    printf("Open\n");

    FILE *fp1 = fopen(wrAlphabet, "w");
    stat(wrAlphabet, &sb);
    printf("\tFp1:\tinode - %d\tsize - %d\n", (int) sb.st_ino, (int) sb.st_size);

    FILE *fp2 = fopen(wrAlphabet, "w");
    stat(wrAlphabet, &sb);
    printf("\tFp2:\tinode - %d\tsize - %d\n", (int) sb.st_ino, (int) sb.st_size);

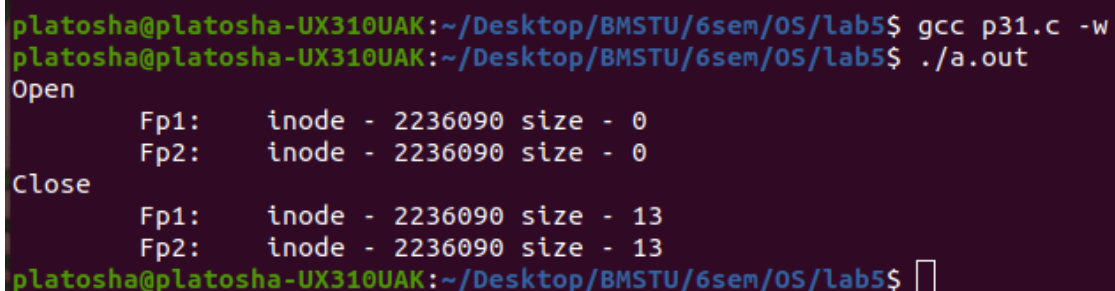
    for (char c = 'a'; c <= 'z'; c++) {
        if (c % 2 == 0) {
            fprintf(fp1, "%c", c);
        }
        else {
            fprintf(fp2, "%c", c);
        }
    }

    printf("Close\n");

    fclose(fp1);
    stat(wrAlphabet, &sb);
    printf("\tFp1:\tinode - %d\tsize - %d\n", (int) sb.st_ino, (int) sb.st_size);

    fclose(fp2);
    stat(wrAlphabet, &sb);
    printf("\tFp2:\tinode - %d\tsize - %d\n", (int) sb.st_ino, (int) sb.st_size);

    return 0;
}
```



```
platosha@platosha-UX310UAK:~/Desktop/BMSTU/6sem/OS/lab5$ gcc p31.c -w
platosha@platosha-UX310UAK:~/Desktop/BMSTU/6sem/OS/lab5$ ./a.out
Open
      Fp1:      inode - 2236090 size - 0
      Fp2:      inode - 2236090 size - 0
Close
      Fp1:      inode - 2236090 size - 13
      Fp2:      inode - 2236090 size - 13
platosha@platosha-UX310UAK:~/Desktop/BMSTU/6sem/OS/lab5$
```

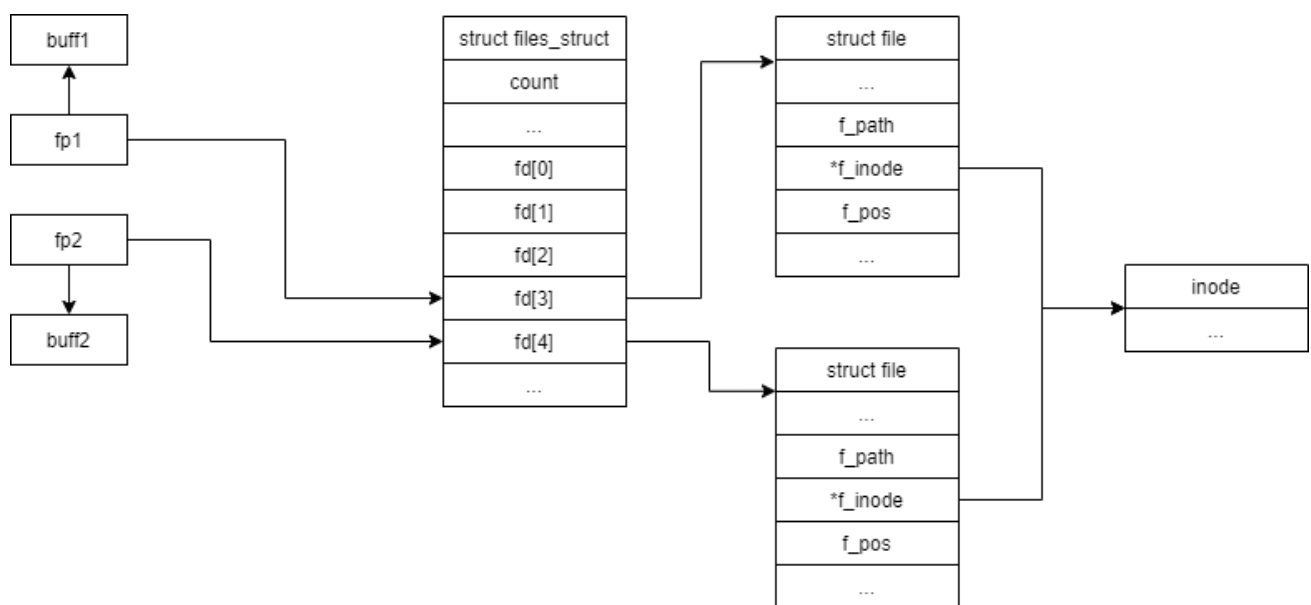
Функция `fopen()` стандартной библиотеки C возвращает указатель на открытый файл. Режим 'r' указывает на открытие файла только для записи;

система помещает указатель в начало файла и обрезает файл до нулевой длины. Если файл не существует - пробует его создать. В результате вызова функции, будут созданы два потока для ввода. Файловые дескрипторы независимы, следовательно, указатели в файле также независимы.

Функция `fprintf()` – функция буферизованного ввода/вывода.

В цикле происходит запись каждого четного символа в буфер, соответствующий `fp1`; каждого нечетного – `fp2`. Запись в файл из буфера происходит при вызове функции `fclose()`.

Поскольку сначала вызывается `fclose(fp1)`, то буфер, ассоциированный с `fp1` будет записан в файл. При следующем вызове `fclose(fp2)` содержимое файла будет перезаписано, в результате чего в нем окажутся лишь символы из буфера, связанного с `fp2`. Таким образом, данные из первого буфера будут утеряны.



Программа 3 (с потоками)

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

#define wrAlphabet "wrAlphabet.txt"

void *thread_write(int *fd)
{
    struct stat sb;

    FILE *fp = fopen(wrAlphabet, "w");
    stat(wrAlphabet, &sb);
    printf("Open\n\tFp%d:\tinode - %d\tsize - %d\n", *fd, (int) sb.st_ino, (int)
sb.st_size);

    for (char c = 'a'; c <= 'z'; c++) {
        if (c % 2 == 0 && *fd % 2 == 1) {
            fprintf(fp, "%c", c);
        }

        if (c % 2 == 1 && *fd % 2 == 0) {
            fprintf(fp, "%c", c);
        }
    }

    fclose(fp);
    stat(wrAlphabet, &sb);
    printf("Close\n\tFp%d:\tinode - %d\tsize - %d\n", *fd, (int) sb.st_ino, (int)
sb.st_size);
}

int main()
{
    pthread_t thread1, thread2;
    int fd1 = 1, fd2 = 2;

    int stat1 = pthread_create(&thread1, NULL, thread_write, &fd1);
    if (stat1 != 0) {
        printf("Error. Can't create thread 1!\n");
        exit(1);
    }

    int stat2 = pthread_create(&thread2, NULL, thread_write, &fd2);
    if (stat2 != 0) {
        printf("Error. Can't create thread 1!\n");
        exit(1);
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```



```

platosh@platosh-UX310UAK:~/Desktop/BMSTU/6sem/OS/lab5$ gcc p32.c -pthread -w
platosh@platosh-UX310UAK:~/Desktop/BMSTU/6sem/OS/lab5$ ./a.out
Open
    Fp1:    inode - 2236090 size - 0
Open
    Fp2:    inode - 2236090 size - 0
Close
    Fp2:    inode - 2236090 size - 13
Close
    Fp1:    inode - 2236090 size - 13
platosh@platosh-UX310UAK:~/Desktop/BMSTU/6sem/OS/lab5$ █

```

Аналогично программе без потоков, будут созданы два независимых дескриптора. Каждый из потоков будет выполнять запись символов в соответствующий буфер. И в результате вызова `fclose()` дважды, в файле сохранить содержимое того буфера, чей поток завершился последним.

```

struct stat {
    dev_t      st_dev;    /* устройство */
    ino_t      st_ino;    /* inode */
    mode_t     st_mode;   /* режим доступа */
    nlink_t    st_nlink;  /* количество жестких ссылок */
    uid_t      st_uid;    /* идентификатор пользователя-владельца */
    gid_t      st_gid;    /* идентификатор группы-владельца */
    dev_t      st_rdev;   /* тип устройства */
                    /* (если это устройство) */
    off_t      st_size;   /* общий размер в байтах */
    blksize_t  st_blksize; /* размер блока ввода-вывода */
                    /* в файловой системе */
    blkcnt_t   st_blocks; /* количество выделенных блоков */
    time_t     st_atime;  /* время последнего доступа */
    time_t     st_mtime;  /* время последней модификации */
    time_t     st_ctime;  /* время последнего изменения */
};

```

```
typedef struct __sFILE {  
    ...  
    unsigned char *_p;  
    short _flags;  
    short _file;  
    struct __sbuf _bf;  
    ...  
    void *_cookie;  
    ...  
    struct __sbuf _ub;  
    struct __sFILEX *_extra;  
    int _ur;  
    ...  
    unsigned char _ubuf[3];  
    unsigned char _nbuf[1];  
    ...  
    struct __sbuf _lb;  
    int _blksize;  
    fpos_t _offset;  
} FILE;
```