



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»

# Отчёт

## по лабораторной работе № 5

**Название:** Буферизованный и не буферизованный ввод-вывод

**Дисциплина:** Операционные системы

Студент

ИУ7-65Б

(Группа)

Д.О. Склифасовский

(Подпись, дата)

(И.О. Фамилия)

Преподаватель

Н.Ю. Рязанова

(Подпись, дата)

(И.О. Фамилия)

*Москва, 2021*

## Программа 1

```
1 #include <stdio.h>
2 #include <fcntl.h>
3
4 int main()
5 {
6     int fd = open("alphabet.txt", O_RDONLY);
7
8     FILE *fs1 = fdopen(fd, "r");
9     char buff1[20];
10    setvbuf(fs1, buff1, _IOFBF, 20);
11
12    FILE *fs2 = fdopen(fd, "r");
13    char buff2[20];
14    setvbuf(fs2, buff2, _IOFBF, 20);
15
16    int flag1 = 1, flag2 = 2;
17    while (flag1 == 1 || flag2 == 1)
18    {
19        char c;
20        flag1 = fscanf(fs1, "%c", &c);
21        if (flag1 == 1)
22        {
23            fprintf(stdout, "%c", c);
24        }
25        flag2 = fscanf(fs2, "%c", &c);
26        if (flag2 == 1)
27        {
28            fprintf(stdout, "%c", c);
29        }
30    }
31    return 0;
32 }
```

```

> ./Part1.out
aubvcwdxeyfzghijklmnopqrst%
~/BMSTU/OS/Lab5

```

С помощью системного вызова `open()` создается дескриптор открытого на чтение файла. Системный вызов `open()` возвращает индекс в массиве `fd` структуры `files_struct`.

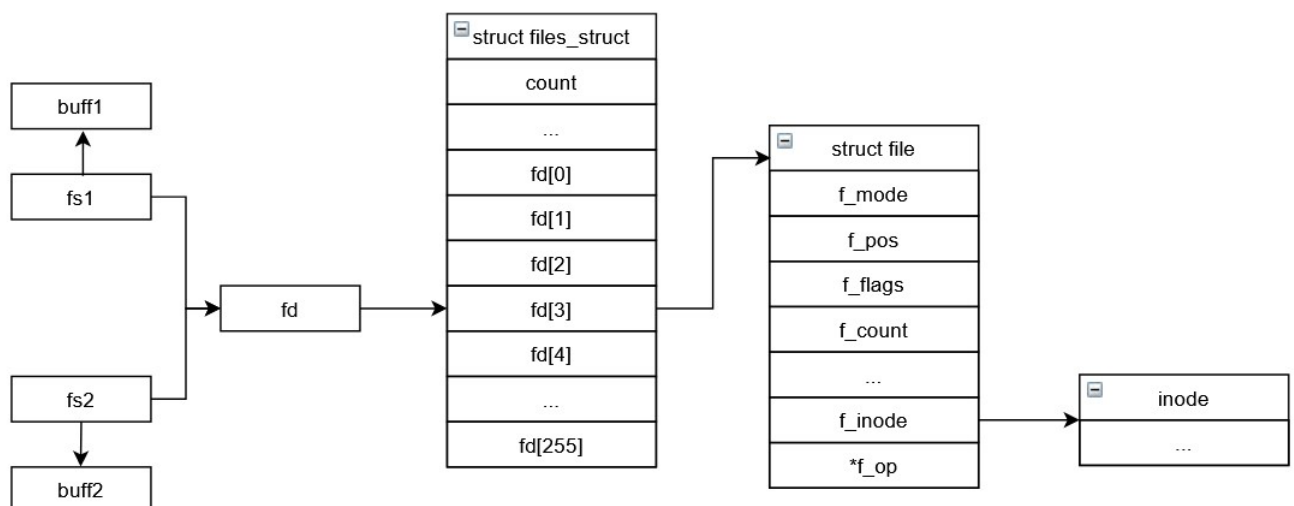
`fdopen()` создает структуру типа `FILE`, которая ссылается на дескриптор, созданный системным вызовом `open`.

Система создаст буфер на 20 байт.

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

Далее в цикле выполняется `fscanf()` поочередно для `fs1` и `fs2`. Так как установлена полная буферизация, то при первом вызове `fscanf()` буфер будет заполнен полностью либо вплоть до конца файла, а `f_pos` установится на следующий за последним записанным в буфер символ.

Результатом является строка "Aubvcwdxeyfzghijklmnopqrst"



## Программа 2

```
1 #include <fcntl.h>
2
3 int main()
4 {
5     char c;
6
7     int fd1 = open("alphabet.txt", O_RDONLY);
8     int fd2 = open("alphabet.txt", O_RDONLY);
9
10    while (read(fd1, &c, 1) == 1 && read(fd2, &c, 1) == 1)
11    {
12        write(1, &c, 1);
13        write(1, &c, 1);
14    }
15    return 0;
16 }
```

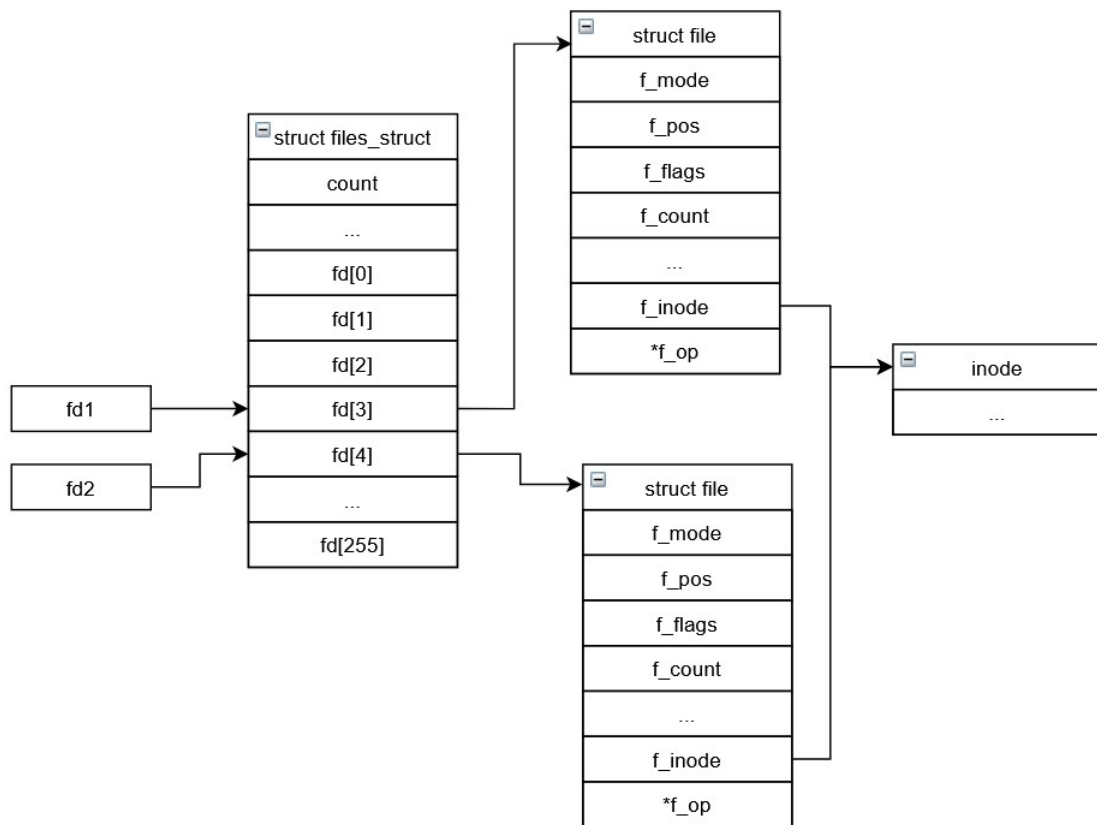


```
> ./Part2.out
aabbccddeeffgghhiijjkkllmmnnnooppqrrssttuuvvwxxyyzz
~/BMSTU/OS/Lab5
```

При вызове системного вызова `open()` создается дескриптор файла в системной таблице файлов, открытых процессом и запись в системной таблице открытых файлов.

файл открывается 2 раза, поэтому в таблице открытых файлов будет 2 дескриптора и каждый такой дескриптор имеет собственный `f_pos`.

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



## Программа 2 с потоками

```

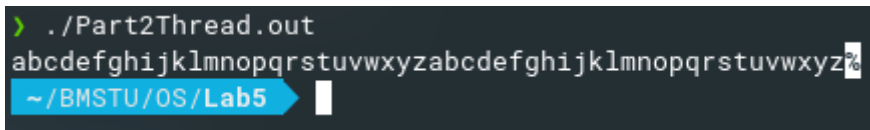
1 #include <fcntl.h>
2 #include <pthread.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 void *read_file(int *fd)
7 {
8     char c;
9     while (read(*fd, &c, 1) == 1)
10     {
11         write(1, &c, 1);
12     }
13     return 0;
14 }
15
16 int main()

```

```

17 {
18     int fd1 = open("alphabet.txt", O_RDONLY);
19     int fd2 = open("alphabet.txt", O_RDONLY);
20
21     pthread_t thread1, thread2;
22
23     int stat1 = pthread_create(&thread1, NULL, read_file, &fd1);
24     if (stat1 != 0)
25     {
26         printf("Cannot create thread 1\n");
27         return -1;
28     }
29
30     int stat2 = pthread_create(&thread2, NULL, read_file, &fd2);
31     if (stat2 != 0)
32     {
33         printf("Cannot create thread 2\n");
34         return -1;
35     }
36
37     pthread_join(thread1, NULL);
38     pthread_join(thread2, NULL);
39
40     return 0;
41 }

```



```

> ./Part2Thread.out
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz%
~/BMSTU/OS/Lab5

```

В данном случае потоки читают один и тот же файл параллельно, каждый поток выполнит запись независимо, что приведет к повтору каждого символа.

### Программа 3

```
1 #include <stdio.h>
2 #include <sys/stat.h>
3
4 int main()
5 {
6     struct stat bufStat;
7
8     FILE *file1 = fopen("result.txt", "w");
9     stat("result.txt", &bufStat);
10    printf("First opening\n\tinode\t= %ld\n\tsize\t= %ld\n",
11           bufStat.st_ino, bufStat.st_size);
12
13    FILE *file2 = fopen("result.txt", "w");
14    stat("result.txt", &bufStat);
15    printf("Second opening\n\tinode\t= %ld\n\tsize\t= %ld\n",
16           bufStat.st_ino, bufStat.st_size);
17
18    char needChar = 'a';
19    while (needChar <= 'z')
20    {
21        if (needChar % 2 == 0)
22        {
23            fprintf(file1, "%c", needChar);
24        }
25        else
26        {
27            fprintf(file2, "%c", needChar);
28        }
29        needChar++;
30    }
31
32    fclose(file1);
33    stat("result.txt", &bufStat);
```

```

32     printf("First closing\n\tinode\t= %ld\n\tsize\t= %ld\n",
        bufStat.st_ino, bufStat.st_size);
33
34     fclose(file2);
35     stat("result.txt", &bufStat);
36     printf("Second closing\n\tinode\t= %ld\n\tsize\t= %ld\n",
        bufStat.st_ino, bufStat.st_size);
37
38     return 0;
39 }

```

```

> ./Part3.out
First opening
    inode    = 2134638
    size     = 0
Second opening
    inode    = 2134638
    size     = 0
First closing
    inode    = 2134638
    size     = 13
Second closing
    inode    = 2134638
    size     = 13

```

Файл открывается 2 раза для записи. Создается два дескриптора открытых файлов, следовательно, два независимых `f_pos`. Но они ссылаются на один `inode`.

`fopen` и `fprintf` функции `stdio` – библиотеки буферизуемого ввода/вывода, поэтому сначала информация пишется в буфер.

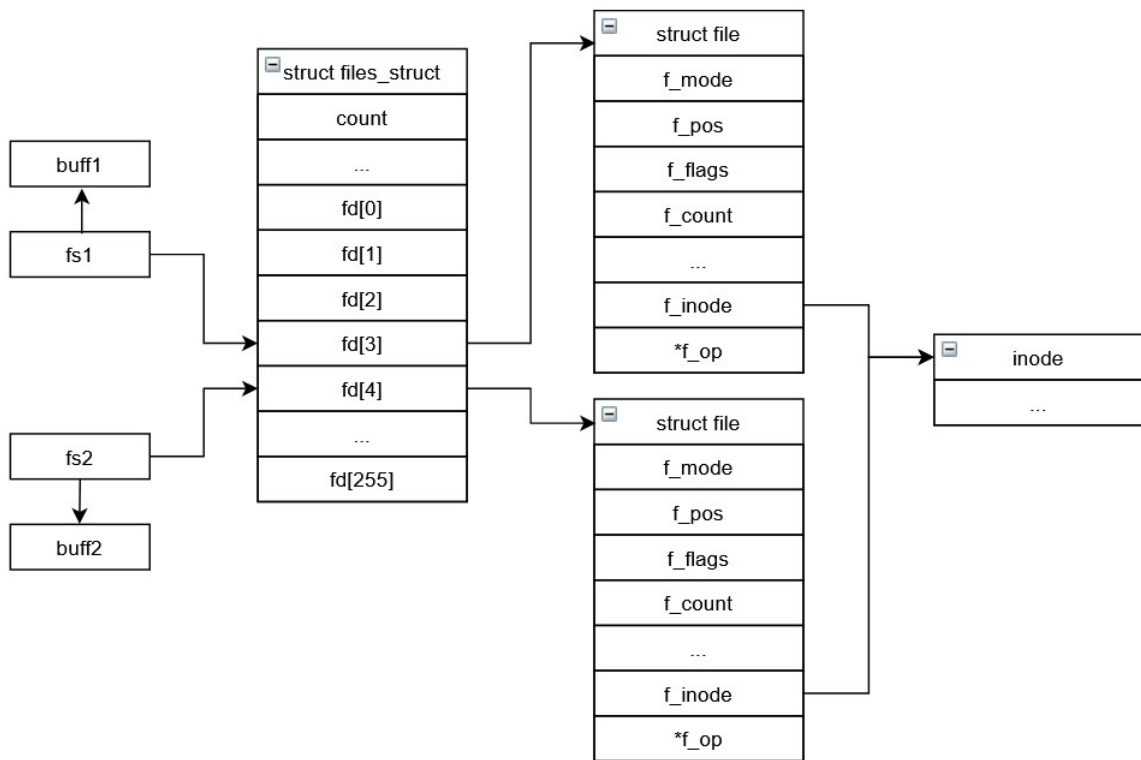
В икле происходит запись каждого четного символа в буфер, соответствующий `file1` и каждого нечетного в буфер, соответствующий `file2`. Запись в файл из буфера происходит при вызове функции `fclose()`.

При вызове `fclose()` для `file1` буфер записывается в файл.

При вызове `fclose()` для `file2`, все содержимое файла перезаписывается содержимым буфера для `file2`.

В итоге, данные из первого буфера будут утеряны.





### Программ 3 с потоками

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/stat.h>
4 #include <pthread.h>
5
6 void *write(int *fd)
7 {
8     struct stat bufStat;
9
10    FILE *file = fopen("result.txt", "w");
11    stat("result.txt", &bufStat);
12    printf("Opening file\n\tinode\t= %ld\n\tsize\t= %ld\n",
13          bufStat.st_ino, bufStat.st_size);
14
15    char needChar = 'a' + *fd - 1;
16    while (needChar <= 'z')
17    {

```

```

17         fprintf(file , "%c" , needChar);
18         needChar += 2;
19     }
20
21     fclose(file);
22     stat("result.txt" , &bufStat);
23     printf("Closing\n\tinode\t= %ld\n\tsize\t= %ld\n" , bufStat.
        st_ino , bufStat.st_size);
24 }
25
26 int main()
27 {
28     pthread_t thread1 , thread2;
29     int fd1 = 1,
30     fd2 = 2;
31
32     int stat1 = pthread_create(&thread1 , NULL, write , &fd1);
33     if (stat1 != 0)
34     {
35         printf("Cannot create thread 1\n");
36         return -1;
37     }
38
39     int stat2 = pthread_create(&thread2 , NULL, write , &fd2);
40     if (stat2 != 0)
41     {
42         printf("Cannot create thread 2\n");
43         return -1;
44     }
45
46     pthread_join(thread1 , NULL);
47     pthread_join(thread2 , NULL);
48     return 0;
49 }

```

```

> ./Part3Thread.out
Opening file
    inode   = 2134638
    size    = 0
Opening file
    inode   = 2134638
    size    = 0
Closing
    inode   = 2134638
    size    = 13
Closing
    inode   = 2134638
    size    = 13
~/BMSTU/OS/Lab5

```

В данном случае потоки записывают символ в соответствующий буфер. В результате вызова `fclose`, в файле сохранится содержимое буфера, чей поток завершится 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;

```