



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»

НАПРАВЛЕНИЕ ПОДГОТОВКИ _____ «09.03.04 Программная инженерия»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6

Название: _____ Буферизованный и не буферизованный ввод-вывод

Дисциплина: _____ Операционные системы

Студент	<u>ИУ7-66Б</u>	_____	<u>А.Д. Ковель</u>
	Группа	Подпись, дата	И. О. Фамилия

Преподаватель	_____	<u>Н. Ю. Рязанова</u>
	Подпись, дата	И. О. Фамилия

Москва, 2023 г.

СТРУКТУРА FILE

```
1 typedef struct _IO_FILE FILE;
2 struct _IO_FILE
3 {
4     int _flags;          /* High-order word is _IO_MAGIC; rest is flags. */
5
6     /* The following pointers correspond to the C++ streambuf protocol. */
7     char *_IO_read_ptr;  /* Current read pointer */
8     char *_IO_read_end;  /* End of get area. */
9     char *_IO_read_base; /* Start of putback+get area. */
10    char *_IO_write_base; /* Start of put area. */
11    char *_IO_write_ptr;  /* Current put pointer. */
12    char *_IO_write_end;  /* End of put area. */
13    char *_IO_buf_base;   /* Start of reserve area. */
14    char *_IO_buf_end;    /* End of reserve area. */
15
16    /* The following fields are used to support backing up and undo. */
17    char *_IO_save_base; /* Pointer to start of non-current get area. */
18    char *_IO_backup_base; /* Pointer to first valid character of backup area */
19    char *_IO_save_end; /* Pointer to end of non-current get area. */
20
21    struct _IO_marker *_markers;
22
23    struct _IO_FILE *_chain;
24
25    int _fileno;
26    int _flags2;
27    __off_t _old_offset; /* This used to be _offset but it's too small. */
28
29    /* 1+column number of pbase(); 0 is unknown. */
30    unsigned short _cur_column;
31    signed char _vtable_offset;
32    char _shortbuf[1];
33
34    _IO_lock_t *_lock;
35    #ifdef _IO_USE_OLD_IO_FILE
36 };
```

ПРОГРАММА №1

```

1 #include <fcntl.h>
2 #include <stdio.h>
3
4 int main() {
5     // have kernel open connection to file alphabet.txt
6     int fd = open("alphabet.txt", O_RDONLY);
7
8     // create two a C I/O buffered streams using the above connection
9     FILE *fs1 = fdopen(fd, "r");
10    char buff1[20];
11    setvbuf(fs1, buff1, _IOFBF, 20);
12
13    FILE *fs2 = fdopen(fd, "r");
14    char buff2[20];
15    setvbuf(fs2, buff2, _IOFBF, 20);
16
17    // read a char & write it alternately from fs1 and fs2
18    int flag1 = 1, flag2 = 2;
19    while (flag1 == 1 || flag2 == 1) {
20        char c;
21
22        flag1 = fscanf(fs1, "%c", &c);
23        if (flag1 == 1) fprintf(stdout, "%c", c);
24
25        flag2 = fscanf(fs2, "%c", &c);
26        if (flag2 == 1) fprintf(stdout, "%c", c);
27    }
28
29    return 0;
30 }

```

Результат работы программы:

```

~/Documents/BMSTU/bmstu-os-6sem/lab-05/code (main*) » ./a.out
aubvcwdxeyfzghijklmnopqrst%

```

Программа с дополнительным потоком:

```

1 #include <fcntl.h>
2 #include <pthread.h>
3 #include <stdio.h>
4
5 void *thread_routine(void *fd) {
6     int flag = 1;
7     char c;
8
9     FILE *fs = fdopen(*((int *)fd), "r");
10    char buf[20];
11    setvbuf(fs, buf, _IOFBF, 20);
12
13    while (flag == 1) {
14        flag = fscanf(fs, "%c", &c);
15        if (flag == 1) {
16            fprintf(stdout, "%c\n", c);
17        }
18    }
19 }
20
21 int main(void) {
22     int fd = open("alphabet.txt", O_RDONLY);
23
24     FILE *fs = fdopen(fd, "r");
25     char buf[20];
26     setvbuf(fs, buf, _IOFBF, 20);
27
28     pthread_t thr_worker;
29
30     pthread_create(&thr_worker, NULL, thread_routine, &fd);
31
32     int flag = 1;
33     char c;
34     while (flag == 1) {
35         flag = fscanf(fs, "%c", &c);
36         if (flag == 1) {
37             fprintf(stdout, "%c\n", c);
38         }
39     }
40     pthread_join(thr_worker, NULL);
41     return 0;
42 }

```

Результат работы программы:

```

~/Documents/BMSTU/bmstu-os-6sem/lab-05/code (main*) » ./a.out
abcdefghijklmnopqrstuvwxyz%

```

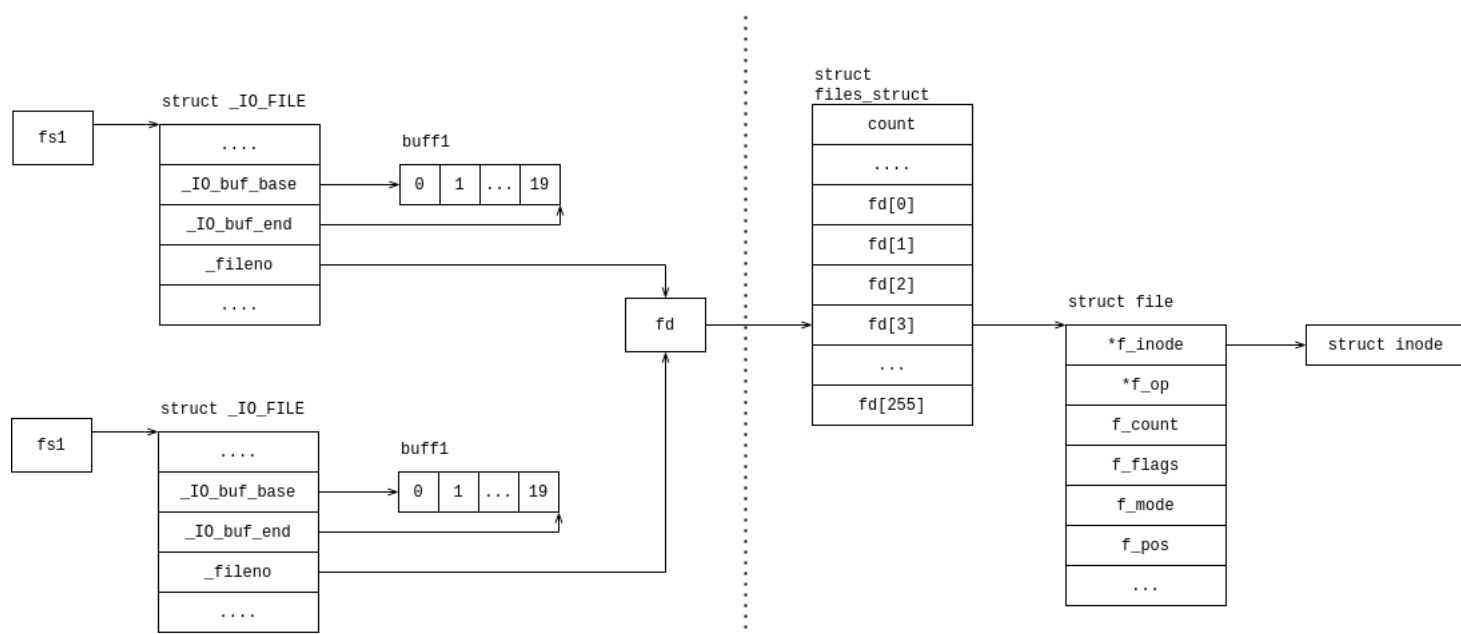
С помощью системного вызова `open()` создается дескриптор открытого только для чтения файла. Системный вызов `open()` возвращает индекс в массиве `fd` структуры `files_struct`. `fdopen()` создает экземпляры структуры типа `FILE` (`fs1` и `fs2`), которые ссылаются на дескриптор, созданный системным вызовом `open`. Далее создаются буферы `buff1` и `buff2` размером 20 байт. Для дескрипторов `fs1` и `fs2` помощью `setbuf` задаются соответствующие буферы и тип буферизации `_IOFBF` (полная буферизация).

Далее `fscanf()` выполняется в цикле поочерёдно 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.

При первом вызове `fscanf(fs1, "%c &c);` в буфер `buff1` считываются первые 20 символов (`abcdefghijklmnopqrst`), в переменную `s` записывается, а затем выводится с помощью `fprintf`, символ `'a'`. При первом вызове `fscanf(fs2, "%c &c);`, в буфер `buff2` считываются оставшиеся в файле символы – `vwxyz` (в переменную `s` записывается символ `'u'`).

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.

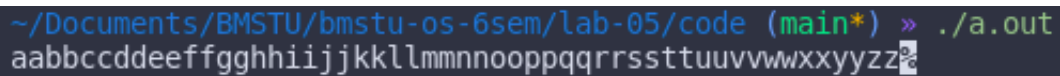
Связь структур:



ПРОГРАММА №2

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

Результат работы программы:



```
~/Documents/BMSTU/bmstu-os-6sem/lab-05/code (main*) >> ./a.out
aabbccddeeffgghhiijjkkllmmnnooppqrrssttuuvvwxxyyzz
```

Программа с дополнительным потоком:

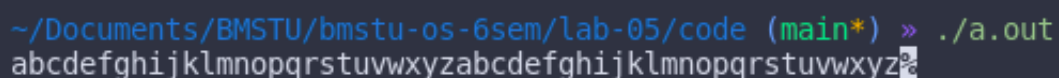
```
1 #include <fcntl.h>
2 #include <pthread.h>
3 #include <stdio.h>
4 #include <unistd.h>
5
6 pthread_mutex_t mux;
7
8 void *thread_routine(void *arg) {
9     int fd = *((int *)arg);
10
11     int flag = 1;
12     char c;
13
14     pthread_mutex_lock(&mux);
15     while (flag == 1) {
16         flag = read(fd, &c, 1);
17         if (flag == 1) write(1, &c, 1);
18     }
```

```

19 pthread_mutex_unlock(&mutex);
20 }
21
22 int main() {
23     int fd1 = open("alphabet.txt", O_RDONLY);
24     int fd2 = open("alphabet.txt", O_RDONLY);
25
26     pthread_t thr_worker;
27
28     if (pthread_mutex_init(&mutex, NULL) != 0) {
29         printf("can't pthread_mutex_init.\n");
30         return 1;
31     }
32
33     pthread_create(&thr_worker, NULL, thread_routine, &fd1);
34
35     int flag = 1;
36     char c;
37
38     pthread_mutex_lock(&mutex);
39     while (flag == 1) {
40         flag = read(fd2, &c, 1);
41         if (flag == 1) write(1, &c, 1);
42     }
43     pthread_mutex_unlock(&mutex);
44
45     pthread_join(thr_worker, NULL);
46
47     return 0;
48 }

```

Результат работы программы:



```

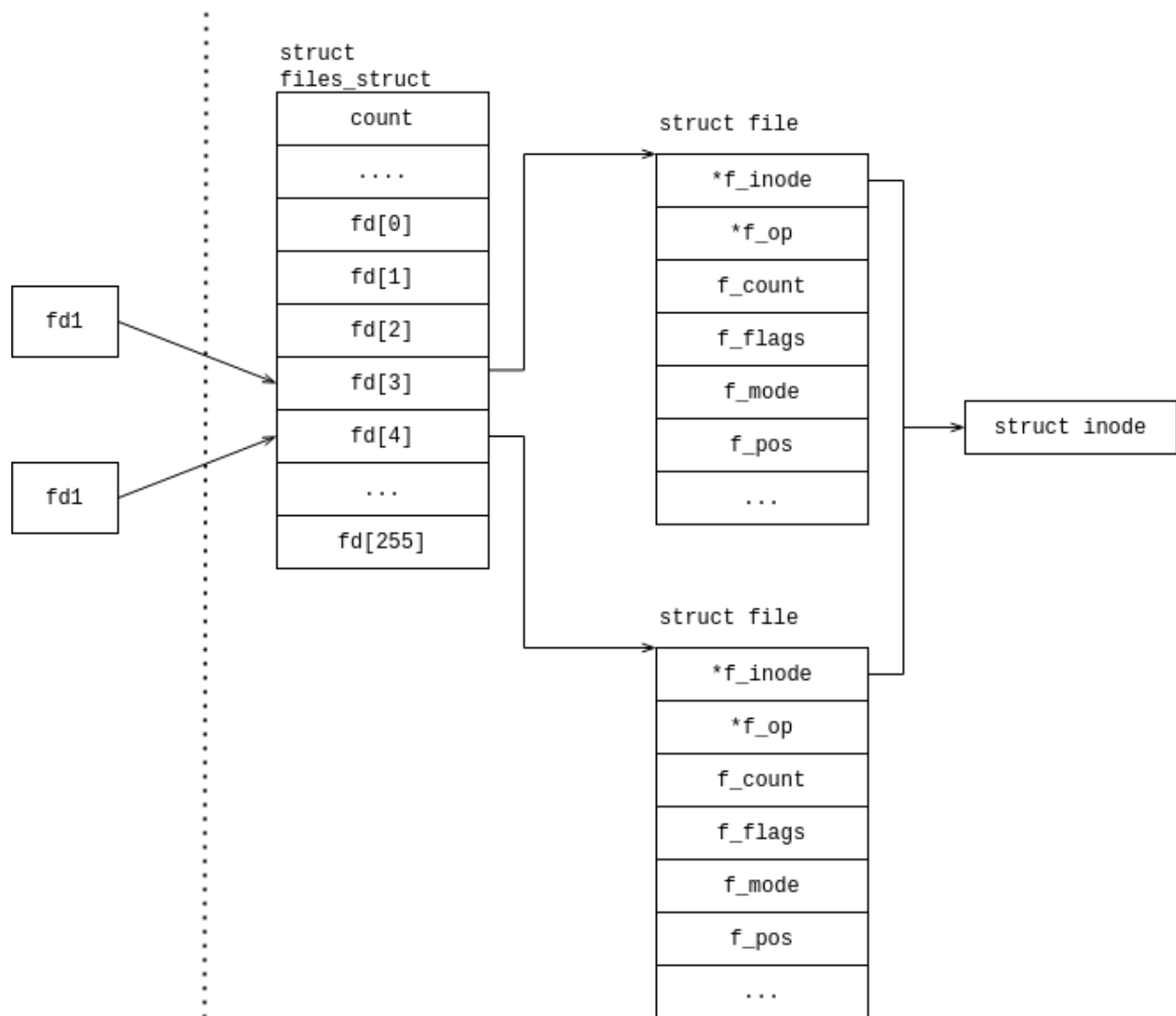
~/Documents/BMSTU/bmstu-os-6sem/lab-05/code (main*) » ./a.out
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz

```

В программе один и тот же файл открыт 2 раза для чтения. При вызове системного вызова `open()` создается дескриптор файла в системной таблице файлов, открытых процессом и запись в системной таблице открытых файлов. Так как в данном случае файл открывается 2 раза, то в таблице открытых процессом файлов будет 2 дескриптора и каждый такой дескриптор имеет собственный `f_pos`. По этой причине чтение становится независимым – при вызове `read()` для обоих дескрипторов по очереди, оба указателя проходят по всем позициям файла, и каждый символ считывается и выводится по два раза. Несмотря на то, что су-

существует 2 дескриптора открытого файла, открывается один и тот же файл, т.е. inode один и тот же.

Связь структур:



ПРОГРАММА №3

Написать программу, которая открывает один и тот же файл два раза с использованием библиотечной функции `open()`. Для этого объявляются два файловых дескриптора. В цикле записать в файл буквы латинского алфавита поочередно передавая функции `fputc()` то первый дескриптор, то – второй. Результат прокомментировать.

```
1 #include <fcntl.h>
2 #include <stdio.h>
3 #include <unistd.h>
4 int main() {
```



```

5 FILE *f1 = fopen("out.txt", "w");
6 FILE *f2 = fopen("out.txt", "w");
7 for (char letr = 'a'; letr < '{'; letr++) {
8     letr % 2 ? fprintf(f1, "%c", letr) : fprintf(f2, "%c", letr);
9 }
10 fclose(f2);
11 fclose(f1);
12 return 0;
13 }

```

Результат работы программы:

```

~/Documents/BMSTU/bmstu-os-6sem/lab-05/code (main*) » ./a.out
~/Documents/BMSTU/bmstu-os-6sem/lab-05/code (main*) » cat out.txt
acegikmoqsuw%

```

Файл out.txt открывается функцией `fopen()` на запись дважды. Создается два дескриптора открытых файлов, две независимые позиции, но с одним и тем же inode. Функция `fprintf()` (функция записи в файл) самостоятельно создаёт буфер, в который заносимая в файл информация первоначально и помещается. Из буфера информация переписывается в результате трех действий:

1. Информация из буфера записывается в файл когда буфер полон. В этом случае содержимое буфера автоматически переписывается в файл.
2. Если вызван `fflush` - принудительная запись содержимого в файл.
3. Если вызван `fclose`.

В данном случае запись в файл происходит в результате вызова функции `fclose`. При вызове `fclose()` для `fs1` буфер для `fs1` записывается в файл. При вызове `fclose()` для `fs2`, все содержимое файла очищается, а в файл записывается содержимое буфера для `fs2`. В итоге произошла утеря данных, в файле окажется только содержимое буфера для `fs2`.

Решение. Необходимо использовать `open()` с флагом `O_APPEND`. Если этот флаг установлен, то каждой операции добавления гарантируется неделимость. Программа с дополнительным потоком:

```

1 #include <fcntl.h>
2 #include <pthread.h>
3 #include <stdio.h>
4 #include <sys/stat.h>
5 #include <unistd.h>
6
7 struct stat statbuf;
8
9 void *thread_routine() {
10     FILE *f2 = fopen("out.txt", "w");
11     stat("out.txt", &statbuf);
12     printf("open for fs2: inode = %ld, bufsizе = %ld blocksize= %ld\n",
13           (long int)statbuf.st_ino, (long int)statbuf.st_size,
14           (long int)statbuf.st_blksize);
15     for (char letr = 'a'; letr < '{'; letr += 2) fprintf(f2, "%c", letr);
16     fclose(f2);
17     stat("out.txt", &statbuf);
18     printf("close for fs2: inode = %ld, bufsizе = %ld blocksize= %ld\n",
19           (long int)statbuf.st_ino, (long int)statbuf.st_size,
20           (long int)statbuf.st_blksize);
21 }
22
23 int main() {
24     FILE *f1 = fopen("out.txt", "w");
25     stat("out.txt", &statbuf);
26     printf("open for fs1: inode = %ld, bufsizе = %ld blocksize= %ld\n",
27           (long int)statbuf.st_ino, (long int)statbuf.st_size,
28           (long int)statbuf.st_blksize);
29
30     pthread_t thr_worker;
31     pthread_create(&thr_worker, NULL, thread_routine, f1);
32     pthread_join(thr_worker, NULL);
33
34     for (char letr = 'a'; letr < '{'; letr += 2) fprintf(f1, "%c", letr);
35
36     fclose(f1);
37     stat("out.txt", &statbuf);
38     printf("close for fs2: inode = %ld, bufsizе = %ld blocksize= %ld\n",
39           (long int)statbuf.st_ino, (long int)statbuf.st_size,
40           (long int)statbuf.st_blksize);
41
42     return 0;
43 }

```

Результат работы программы:

```
~/Documents/BMSTU/bmstu-os-6sem/lab-05/code (main*) » ./a.out
open for fs1: inode = 814681, bufsize = 0 blocksize= 4096
open for fs2: inode = 814681, bufsize = 0 blocksize= 4096
close for fs2: inode = 814681, bufsize = 13 blocksize= 4096
close for fs2: inode = 814681, bufsize = 13 blocksize= 4096
```

```
~/Documents/BMSTU/bmstu-os-6sem/lab-05/code (main*) » cat out.txt
acegikmoqsuwy%
```

Связь структур:

