



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 дисциплине «Операционные системы»

Тема Буферизованный и не буферизованный ввод-вывод

Студент Зайцева А. А.

Группа ИУ7-62Б

Оценка (баллы) _____

Преподаватель Рязанова Н. Ю.

Москва — 2022 г.

Структура `_IO_FILE`

Листинг 1: Листинг структуры `_IO_FILE`

```
1 // /usr/include/x86_64-linux-gnu/bits/types/FILE.h:
2
3 #ifndef __FILE_defined
4 #define __FILE_defined 1
5
6 struct _IO_FILE;
7
8 /* The opaque type of streams. This is the definition used
   elsewhere. */
9 typedef struct _IO_FILE FILE;
10
11 #endif
12
13
14 // /usr/include/x86_64-linux-gnu/bits/libio.h:
15 ...
16 struct _IO_FILE {
17     int _flags;          /* High-order word is _IO_MAGIC; rest is
   flags. */
18     #define _IO_file_flags _flags
19
20     /* The following pointers correspond to the C++ streambuf protocol
   . */
21     /* Note: Tk uses the _IO_read_ptr and _IO_read_end fields
   directly. */
22     char* _IO_read_ptr;  /* Current read pointer */
23     char* _IO_read_end;  /* End of get area. */
24     char* _IO_read_base; /* Start of putback+get area. */
25     char* _IO_write_base; /* Start of put area. */
26     char* _IO_write_ptr; /* Current put pointer. */
27     char* _IO_write_end; /* End of put area. */
28     char* _IO_buf_base;  /* Start of reserve area. */
29     char* _IO_buf_end;   /* End of reserve area. */
30     /* The following fields are used to support backing up and undo.
   */

```

```

31  char *_IO_save_base; /* Pointer to start of non-current get area.
    */
32  char *_IO_backup_base; /* Pointer to first valid character of
    backup area */
33  char *_IO_save_end; /* Pointer to end of non-current get area. */
34
35  struct _IO_marker *_markers;
36
37  struct _IO_FILE *_chain;
38
39  int _fileno;
40  #if 0
41  int _blksize;
42  #else
43  int _flags2;
44  #endif
45  _IO_off_t _old_offset; /* This used to be _offset but it's too
    small. */
46
47  #define __HAVE_COLUMN /* temporary */
48  /* 1+column number of pbase(); 0 is unknown. */
49  unsigned short _cur_column;
50  signed char _vtable_offset;
51  char _shortbuf[1];
52
53  /* char* _save_gptr; char* _save_egptr; */
54
55  _IO_lock_t *_lock;
56  #ifdef _IO_USE_OLD_IO_FILE
57  };
58  ...

```

1 Первая программа

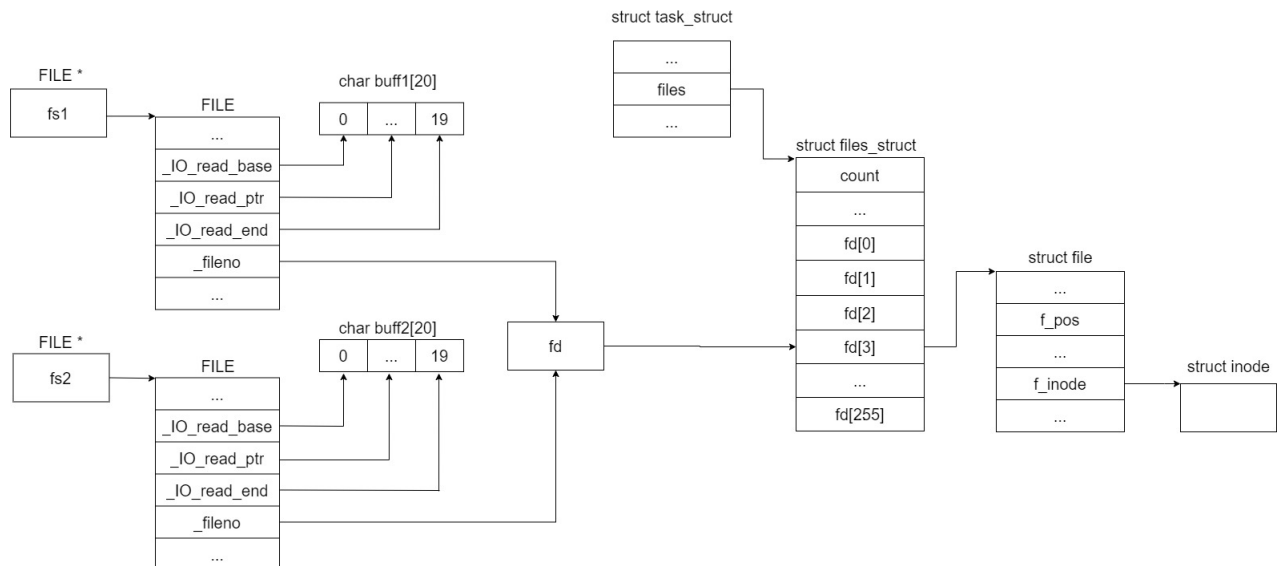


Рис. 1.1: Созданные дескрипторы и связь между ними для первой программы

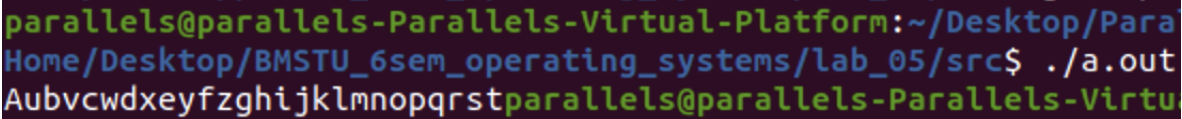
Листинг 1.1: Первая программа

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4
5 int main(void)
6 {
7     // have kernel open connection to file alphabet.txt
8     int fd = open("alphabet.txt", O_RDONLY);
9
10    // create two C I/O buffered streams using the above connection
11    FILE *fs1 = fdopen(fd, "r");
12    char buff1[20];
13    setvbuf(fs1, buff1, _IOFBF, 20);
14
15    FILE *fs2 = fdopen(fd, "r");
16    char buff2[20];
17    setvbuf(fs2, buff2, _IOFBF, 20);
18
19 }
```

```

20     int flag1 = 1, flag2 = 2;
21     while (flag1 == 1 || flag2 == 1)
22     {
23         char c;
24
25         if ((flag1 = fscanf(fs1, "%c", &c)) == 1)
26         {
27             fprintf(stdout, "%c", c);
28         }
29         if ((flag2 = fscanf(fs2, "%c", &c)) == 1)
30         {
31             fprintf(stdout, "%c", c);
32         }
33     }
34     close(fd);
35     return 0;
36 }

```



```

parallels@parallels-Parallels-Virtual-Platform:~/Desktop/Para
Home/Desktop/BMSTU_6sem_operating_systems/lab_05/src$ ./a.out
Aubvcwdxeyfzghijklmnopqrstparallels@parallels-Parallels-Virtu

```

Рис. 1.2: Результат работы первой программы

Листинг 1.2: Первая программа (реализация с потоками)

```

1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <pthread.h>
4 #include <sys/types.h>
5 #include <sys/wait.h>
6 #include <unistd.h>
7
8 void *thread_func(void *args)
9 {
10     int flag = 1;
11     FILE *fs = (FILE *)args;
12
13

```

```

14     while (flag == 1)
15     {
16         char c;
17         if ((flag = fscanf(fs, "%c", &c)) == 1)
18         {
19             fprintf(stdout, "Additional thread read: %c\n", c);
20         }
21     }
22 }
23 int main(void)
24 {
25     setbuf(stdout, NULL);
26     pthread_t thread;
27     int fd = open("alphabet.txt", O_RDONLY);
28
29     FILE *fs1 = fdopen(fd, "r");
30     char buff1[20];
31     setvbuf(fs1, buff1, _IOFBF, 20);
32
33     FILE *fs2 = fdopen(fd, "r");
34     char buff2[20];
35     setvbuf(fs2, buff2, _IOFBF, 20);
36
37     if (pthread_create(&thread, NULL, thread_func, (void *)fs2) !=
38         0)
39     {
40         perror("Error in pthread_create\n");
41         return -1;
42     }
43
44     int flag = 1;
45     while (flag == 1)
46     {
47         char c;
48         if ((flag = fscanf(fs1, "%c", &c)) == 1)
49         {
50             fprintf(stdout, "Main thread read: %c\n", c);
51         }
52     }

```

```

52 pthread_join(thread, NULL);
53 close(fd);
54 return 0;
55 }

```

```

Main thread read: A
Main thread read: B
Main thread read: C
Main thread read: D
Main thread read: E
Main thread read: F
Main thread read: G
Main thread read: H
Main thread read: I
Main thread read: J
Main thread read: K
Main thread read: L
Main thread read: M
Main thread read: N
Main thread read: O
Additional thread read: U
Additional thread read: V
Additional thread read: W
Main thread read: P
Main thread read: Q
Main thread read: R
Additional thread read: X
Additional thread read: Y
Additional thread read: Z
Main thread read: S
Main thread read: T

```

Рис. 1.3: Результат работы первой программы (с потоками)

Системный вызов `open()` создает новый файловый дескриптор для открытого только для чтения файла "alphabet.txt" (который содержит символы `Abcdefghijklmnopqrstuvwxyz`), создавая запись в системной таблице открытых файлов. Эта запись регистрирует смещение в файле и флаги состояния файла. Дескриптор файла является ссылкой на одну из этих записей.

Вызов `fdopen()` возвращает указатель на структуру типа `FILE` (`fs1` и `fs2`), которая ссылается на дескриптор открытого файла, созданный ранее.

Вызов функции `setvbuf()` (для `fs1` и `fs2`) явно задает буффер и его размер (20 байт) и меняет тип буферизации на полную (`_IOFBF`).

При первом вызове `fscanf()` (для `fs1`) буффер `buff1` полностью заполнится первыми 20 символами. Значение `f_pos` в структуре `struct _file` открытого

файла увеличится на 20. При следующем вызове `fscanf()` (для `fs2`) в `buff2` считаются оставшиеся 6 символов, начиная с `f_pos` (`fs1` и `fs2` ссылаются на один и тот же дескриптор `fd`).

Затем в однопоточной программе в цикле поочередно выводятся символы из `buff1` и `buff2` (так как в `buff2` записались лишь оставшиеся 6 символов, после 6 итерации цикла будут выводиться символы только из `buff1`). В двупоточной программе главный поток начинает вывод быстрее, так как для второго потока сначала затрачивается время на его создание, и только потом начинается вывод.

2 Вторая программа

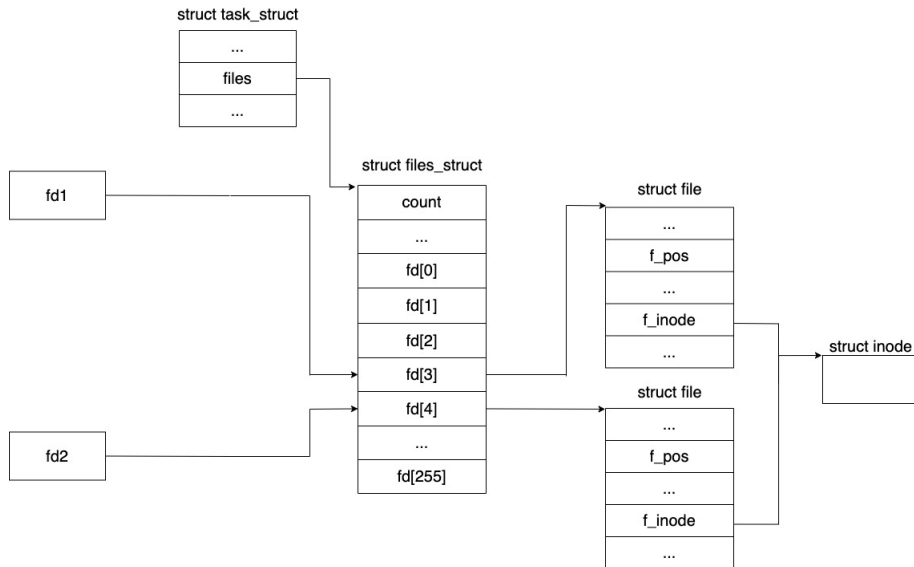


Рис. 2.1: Созданные дескрипторы и связь между ними для второй программы

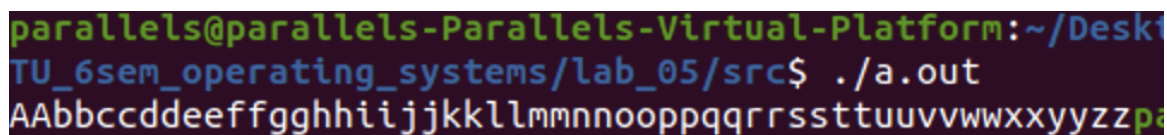
Листинг 2.1: Вторая программа

```
1 #include <fcntl.h>
2 #include <unistd.h>
3
4 int main(void)
5 {
6     int fd1 = open("alphabet.txt", O_RDONLY);
7     int fd2 = open("alphabet.txt", O_RDONLY);
8     int rc1 = 1, rc2 = 1;
9
10    while (rc1 == 1 && rc2 == 1)
11    {
12        char c;
13
14        rc1 = read(fd1, &c, 1);
15        if (rc1 == 1)
16        {
17            write(1, &c, 1);
18            rc2 = read(fd2, &c, 1);
19            if (rc2 == 1)
```

```

20         {
21             write(1, &c, 1);
22         }
23     }
24 }
25 close(fd1);
26 close(fd2);
27 return 0;
28 }

```



```

parallels@parallels-Parallels-Virtual-Platform:~/Desktop/TU_6sem_operating_systems/lab_05/src$ ./a.out
AAbbccddeeffgghhiijjkkllmmnnooppqrrssttuuvvwwxxyyzzp

```

Рис. 2.2: Результат работы второй программы

Листинг 2.2: Вторая программа (с потоками)

```

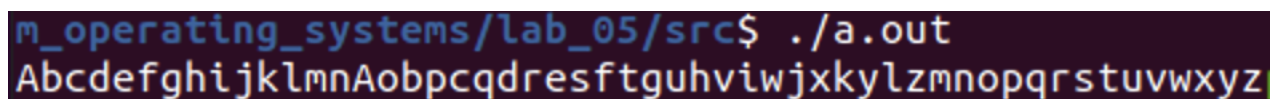
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #include <pthread.h>
5
6 void *thread_func(void *args)
7 {
8     int fd2 = open("alphabet.txt", O_RDONLY);
9     int rc = 1;
10
11     while (rc == 1)
12     {
13         char c;
14         rc = read(fd2, &c, 1);
15         if (rc == 1)
16         {
17             write(1, &c, 1);
18         }
19     }
20     close(fd2);
21 }

```

```

22
23 int main(void)
24 {
25     int fd1 = open("alphabet.txt", O_RDONLY);
26
27     pthread_t thread;
28     if (pthread_create(&thread, NULL, thread_func, 0) != 0)
29     {
30         perror("error in pthread_create\n");
31         return -1;
32     }
33
34     int rc = 1;
35     while (rc == 1)
36     {
37         char c;
38         rc = read(fd1, &c, 1);
39         if (rc == 1)
40         {
41             write(1, &c, 1);
42         }
43     }
44
45     pthread_join(thread, NULL);
46     close(fd1);
47
48     return 0;
49 }

```



```

m_operating_systems/lab_05/src$ ./a.out
AbcdefghijklmnAobpcqdrstguhviwjkxyz

```

Рис. 2.3: Результат работы второй программы (с потоками)

Два системных вызова `open()` создают два новых файловых дескриптора для открытого только для чтения файла, создавая две записи в системной таблице открытых файлов. Каждая запись регистрирует смещение в файле и флаги состояния файла.

Таким образом, в программе существует две различные структуры `struct file`, которые при этом ссылаются на одну и ту же структуру `struct inode`. В каждой структуре свое поле `f_pos` (то есть смещения независимы), поэтому на экран каждый символ выводится дважды.

При этом в однопоточной программе в цикле каждый символ из файла выводится два раза подряд, а в двупоточной заранее предсказать порядок вывода символов невозможно, так как потоки выполняются параллельно (при этом дочерний поток начинает вывод позже, так как затрачивается время на его создание).

3 Третья программа

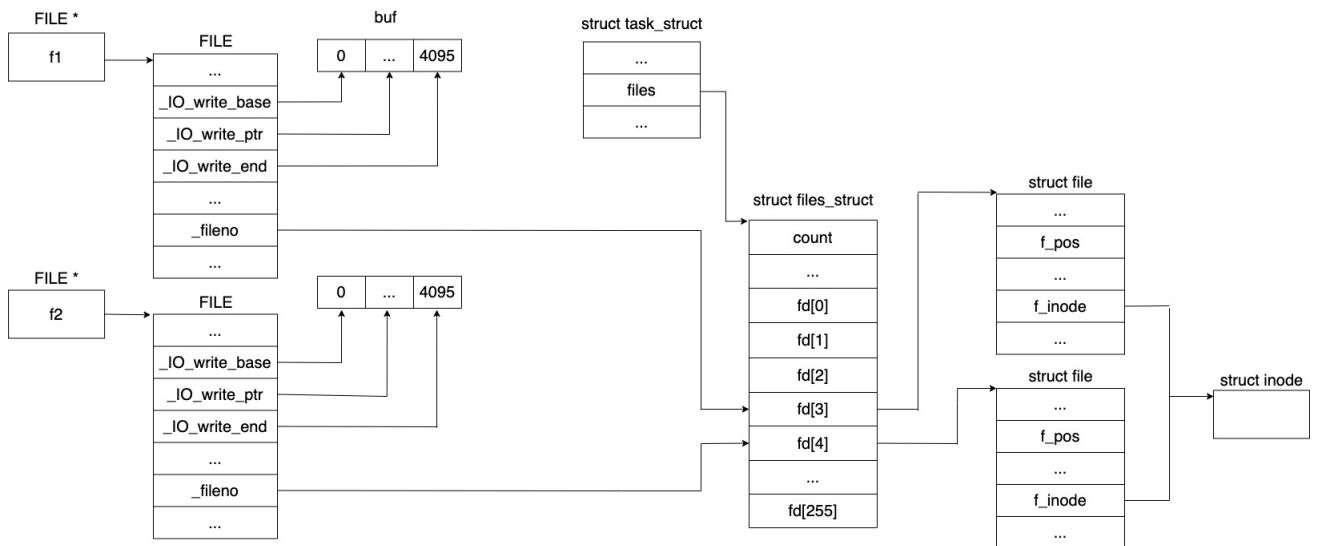


Рис. 3.1: Созданные дескрипторы и связь между ними для третьей программы

Листинг 3.1: Третья программа

```

1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #include <sys/stat.h>
5
6 #define FILENAME "outfile.txt"
7
8 void print_file_info(char *message)
9 {
10     struct stat statbuf;
11     printf("%s", message);
12     if (stat(FILENAME, &statbuf) == 0)
13     {
14         printf("st_ino: %d\n", statbuf.st_ino);
15         printf("st_size: %d\n", statbuf.st_size);
16         printf("st_blksize: %d\n\n", statbuf.st_blksize);
17     }
18     else
19         printf("Error in stat\n\n");
20 }

```

```

21
22 int main()
23 {
24     print_file_info("Before first open\n");
25     FILE *f1 = fopen(FILENAME, "w");
26     print_file_info("After first open\n");
27     FILE *f2 = fopen(FILENAME, "w");
28     print_file_info("After second open\n");
29
30     for (char c = 'a'; c <= 'z'; c++)
31     {
32         if (c % 2)
33         {
34             fprintf(f1, "%c", c);
35         }
36         else
37         {
38             fprintf(f2, "%c", c);
39         }
40     }
41
42     print_file_info("Before first close\n");
43     fclose(f1);
44     print_file_info("After first close\n");
45     fclose(f2);
46     print_file_info("After second close\n");
47
48     return 0;
49 }

```

```
Home/Desktop/BMSTU_6sem_operating_systems/lab_05/src$ ./a.out
Before first open
Error in stat

After first open
st_ino: 24525411
st_size: 0
st_blksize: 4096

After second open
st_ino: 24525411
st_size: 0
st_blksize: 4096

Before first close
st_ino: 24525411
st_size: 0
st_blksize: 4096

After first close
st_ino: 24525411
st_size: 13
st_blksize: 4096

After second close
st_ino: 24525411
st_size: 13
st_blksize: 4096
```

Рис. 3.2: Результат работы третьей программы

Содержимое файла outfile.txt: bdfhjlnprtvxz

Файл outfile.txt дважды открывается на запись с помощью функции `open()`. С помощью функции `fprintf()` стандартной библиотеки `stdio.h` выполняется буферизованный вывод. Буфер создается без явного вмешательства. Информация сначала пишется в буфер, а из буфера информация переписывается в файл, если произошло одно из 3 событий:

1. буфер заполнен;
2. вызвана функция `fclose()` (в данной программе именно эти события приводят к записи в файл);
3. вызвана функция `fflush()` (принудительная запись в файл).

Так как `f_pos` независимы для каждого дескриптора файла, запись в файл в каждом случае в данной программе производится с его начала.

Символы, имеющие четный код в таблице ASCII (b, d, ...) записываются в буфер, который относится к структуре, на которую указывает f2, нечетный (a, c, ...) – к f1.

Данные, которые были записаны после первого вызова fclose (для f1), были потеряны в результате второго вызова fclose (для f2), поэтому в файле outfile.txt записаны только символы bdfhjlnprtvxz (из буфера, относящегося к f2).

Если поменять вызовы fclose для f1 и f2 местами, то результат будет противоположным: acegikmoqsuwu

Листинг 3.2: Третья программа (реализация с потоками)

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <pthread.h>
4 #include <unistd.h>
5 #include <sys/stat.h>
6
7 #define FILENAME "outfile.txt"
8
9 void print_file_info(char *message)
10 {
11     struct stat statbuf;
12     printf("%s", message);
13     if (stat(FILENAME, &statbuf) == 0)
14     {
15         printf("st_ino: %ld\n", statbuf.st_ino);
16         printf("st_size: %ld\n", statbuf.st_size);
17         printf("st_blksize: %ld\n\n", statbuf.st_blksize);
18     }
19     else
20         printf("Error in stat\n\n");
21 }
22
23 void *thread_func(void *args)
24 {
25     FILE *f2 = fopen(FILENAME, "w");
26     print_file_info("After second open\n");
```



```

27
28     for (char c = 'b'; c <= 'z'; c += 2)
29     {
30         fprintf(f2, "%c", c);
31     }
32     print_file_info("Before first close\n");
33     fclose(f2);
34     print_file_info("After first close\n");
35 }
36
37 int main()
38 {
39     print_file_info("Before first open\n");
40     FILE *f1 = fopen(FILENAME, "w");
41     print_file_info("After first open\n");
42
43
44     pthread_t thread;
45     int rc = pthread_create(&thread, NULL, thread_func, NULL);
46
47     for (char c = 'a'; c <= 'z'; c += 2)
48     {
49         fprintf(f1, "%c", c);
50     }
51
52     pthread_join(thread, NULL);
53     fclose(f1);
54     print_file_info("After second close\n");
55
56
57     return 0;
58 }

```

```
parallels@parallels-Parallels-Virtual-Platform:~/Desktop/Parallels
Home/Desktop/BMSTU_6sem_operating_systems/lab_05/src$ ./a.out
Before first open
Error in stat

After first open
st_ino: 24529012
st_size: 0
st_blksize: 4096

After second open
st_ino: 24529012
st_size: 0
st_blksize: 4096

Before first close
st_ino: 24529012
st_size: 0
st_blksize: 4096

After first close
st_ino: 24529012
st_size: 13
st_blksize: 4096

After second close
st_ino: 24529012
st_size: 13
st_blksize: 4096
```

Рис. 3.3: Результат работы третьей программы (с потоками)

Содержимое файла outfile.txt: aсegikmoqsuwy

В двупоточной реализации принцип действий аналогичен (в данном случае теряются данные, связанные с f2, так как для него fclose вызывается раньше, чем для f1).