



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
по курсу
«Операционные системы»

Тема Буферизованный и небуферизованный ввод-вывод

Студент Сироткина П.Ю.

Группа ИУ7-66Б

Преподаватель Рязанова Н.Ю.

Москва — 2022 г.

1. ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

1.1 СТРУКТУРА FILE

Версия ядра: 5.13.0-40-generic.

Структура FILE описана в файле */usr/include/x86_64-linux-gnu/bits/types/FILE.h*.

Листинг 1.1 – Структура FILE

```
1 #ifndef __FILE_defined
2 #define __FILE_defined 1
3
4 struct _IO_FILE;
5
6 /* The opaque type of streams. This is the definition used elsewhere. */
7 typedef struct _IO_FILE FILE;
8
9 #endif
```

1.2 СТРУКТУРА __IO_FILE

Структура `__IO_FILE` описана в файле */usr/include/x86_64-linux-gnu/bits/types/struct_FILE.h*.

Листинг 1.2 – Структура `__IO_FILE`

```
1 struct _IO_FILE
2 {
3     int _flags; /* High-order word is _IO_MAGIC; rest is flags. */
4
5     /* The following pointers correspond to the C++ streambuf protocol. */
6     char *_IO_read_ptr; /* Current read pointer */
7     char *_IO_read_end; /* End of get area. */
8     char *_IO_read_base; /* Start of putback+get area. */
9     char *_IO_write_base; /* Start of put area. */
10    char *_IO_write_ptr; /* Current put pointer. */
11    char *_IO_write_end; /* End of put area. */
12    char *_IO_buf_base; /* Start of reserve area. */
13    char *_IO_buf_end; /* End of reserve area. */
14
15    /* The following fields are used to support backing up and undo. */
16    char *_IO_save_base; /* Pointer to start of non-current get area. */
```

```

17 char *_IO_backup_base; /* Pointer to first valid character of backup area
   */
18 char *_IO_save_end; /* Pointer to end of non-current get area. */
19
20 struct _IO_marker *_markers;
21
22 struct _IO_FILE *_chain;
23
24 int _fileno;
25 int _flags2;
26 __off_t _old_offset; /* This used to be _offset but it's too small. */
27
28 /* 1+column number of pbase(); 0 is unknown. */
29 unsigned short _cur_column;
30 signed char _vtable_offset;
31 char _shortbuf[1];
32
33 _IO_lock_t *_lock;
34 #ifdef _IO_USE_OLD_IO_FILE
35 };
36
37 struct _IO_FILE_complete
38 {
39     struct _IO_FILE _file;
40     #endif
41     __off64_t _offset;
42     /* Wide character stream stuff. */
43     struct _IO_codecvt *_codecvt;
44     struct _IO_wide_data *_wide_data;
45     struct _IO_FILE *_freeres_list;
46     void *_freeres_buf;
47     size_t __pad5;
48     int _mode;
49     /* Make sure we don't get into trouble again. */
50     char _unused2[15 * sizeof (int) - 4 * sizeof (void *) - sizeof (size_t)];
51 };

```

1.3 ПЕРВАЯ ПРОГРАММА

Листинг 1.3 – Программа №1 (один поток)

```
1 #include <fcntl.h>
2 #include <stdio.h>
3
4 #define BUF_SIZE 20
5
6 int main()
7 {
8     int fd = open("data/alphabet.txt", O_RDONLY);
9
10    FILE *fs1 = fdopen(fd, "r");
11    char buff1[BUF_SIZE];
12    setvbuf(fs1, buff1, _IOFBF, BUF_SIZE);
13
14    FILE *fs2 = fdopen(fd, "r");
15    char buff2[BUF_SIZE];
16    setvbuf(fs2, buff2, _IOFBF, BUF_SIZE);
17
18    int flag1 = 1, flag2 = 2;
19    char c;
20    while (flag1 == 1 || flag2 == 1)
21    {
22        flag1 = fscanf(fs1, "%c", &c);
23        if (flag1 == 1)
24        {
25            fprintf(stdout, "%c", c);
26        }
27        flag2 = fscanf(fs2, "%c", &c);
28        if (flag2 == 1)
29        {
30            fprintf(stdout, "%c", c);
31        }
32    }
33
34    fprintf(stdout, "\n");
35
36    return 0;
37 }
```

На рисунке 1.1 представлен результат работы первой программы.

```
polina@polina:~/sem_06/lab_05/src$ gcc prog_01.c -o app.exe
polina@polina:~/sem_06/lab_05/src$ ./app.exe
a b c d e f g h i j k l m n o p q r s t
```

Рисунок 1.1 – Результат работы первой программы (один поток)

Листинг 1.4 – Программа №1 (два потока)

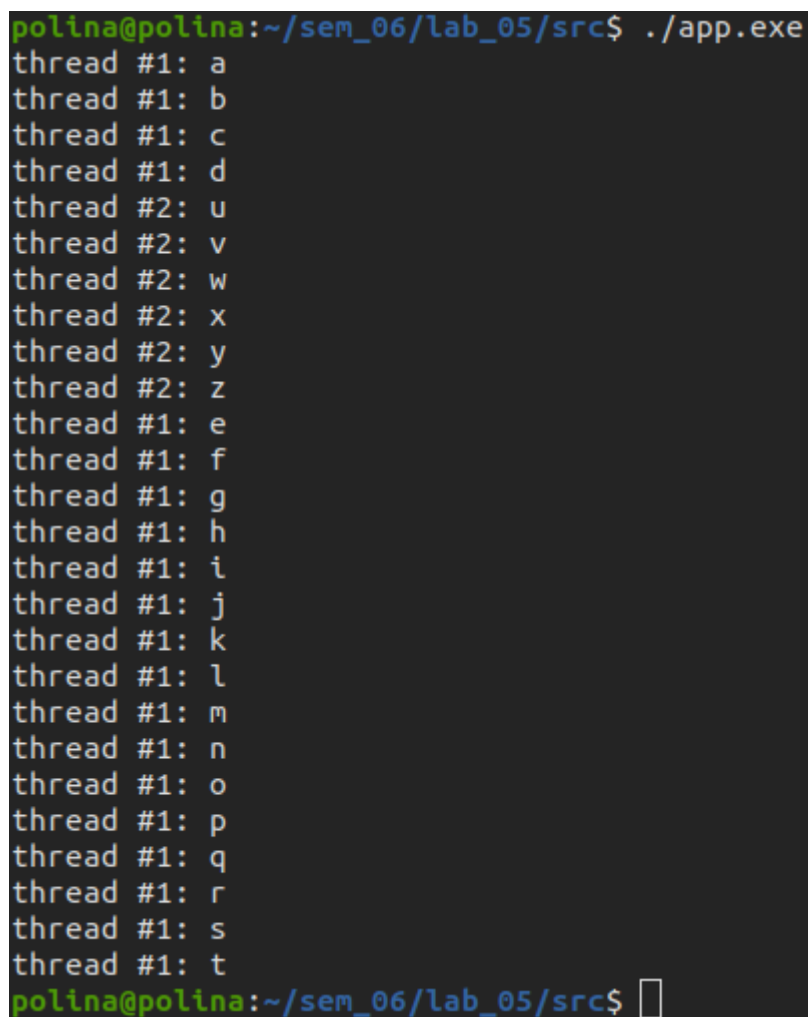
```
1 #include <fcntl.h>
2 #include <stdio.h>
3 #include <unistd.h>
4 #include <pthread.h>
5
6 #define BUF_SIZE 20
7
8 void *thread_run(void *fs)
9 {
10     int flag = 1;
11     char c;
12     while (flag == 1)
13     {
14         flag = fscanf(fs, "%c", &c);
15         if (flag == 1)
16             fprintf(stdout, "thread #2: %c\n", c);
17     }
18     return NULL;
19 }
20
21 int main()
22 {
23     int fd = open("data/alphabet.txt", O_RDONLY);
24     pthread_t td;
25
26     FILE *fs1 = fdopen(fd, "r");
27     char buff1[BUF_SIZE];
28     setvbuf(fs1, buff1, _IOFBF, BUF_SIZE);
29
30     FILE *fs2 = fdopen(fd, "r");
31     char buff2[BUF_SIZE];
32     setvbuf(fs2, buff2, _IOFBF, BUF_SIZE);
33
34     pthread_create(&td, NULL, thread_run, fs2);
35     usleep(1);
```

```

36
37 int flag = 1;
38 char c;
39 while (flag == 1)
40 {
41     flag = fscanf(fs1, "%c", &c);
42     if (flag == 1)
43         fprintf(stdout, "thread #1: %c\n", c);
44 }
45 pthread_join(td, NULL);
46
47 return 0;
48 }

```

На рисунке 1.2 представлен результат работы первой программы.



```

polina@polina:~/sem_06/lab_05/src$ ./app.exe
thread #1: a
thread #1: b
thread #1: c
thread #1: d
thread #2: u
thread #2: v
thread #2: w
thread #2: x
thread #2: y
thread #2: z
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
polina@polina:~/sem_06/lab_05/src$ 

```

Рисунок 1.2 – Результат работы первой программы (два потока)

1.4 АНАЛИЗ

На рисунке 1.3 представлена схема структур, используемых в первой программе.

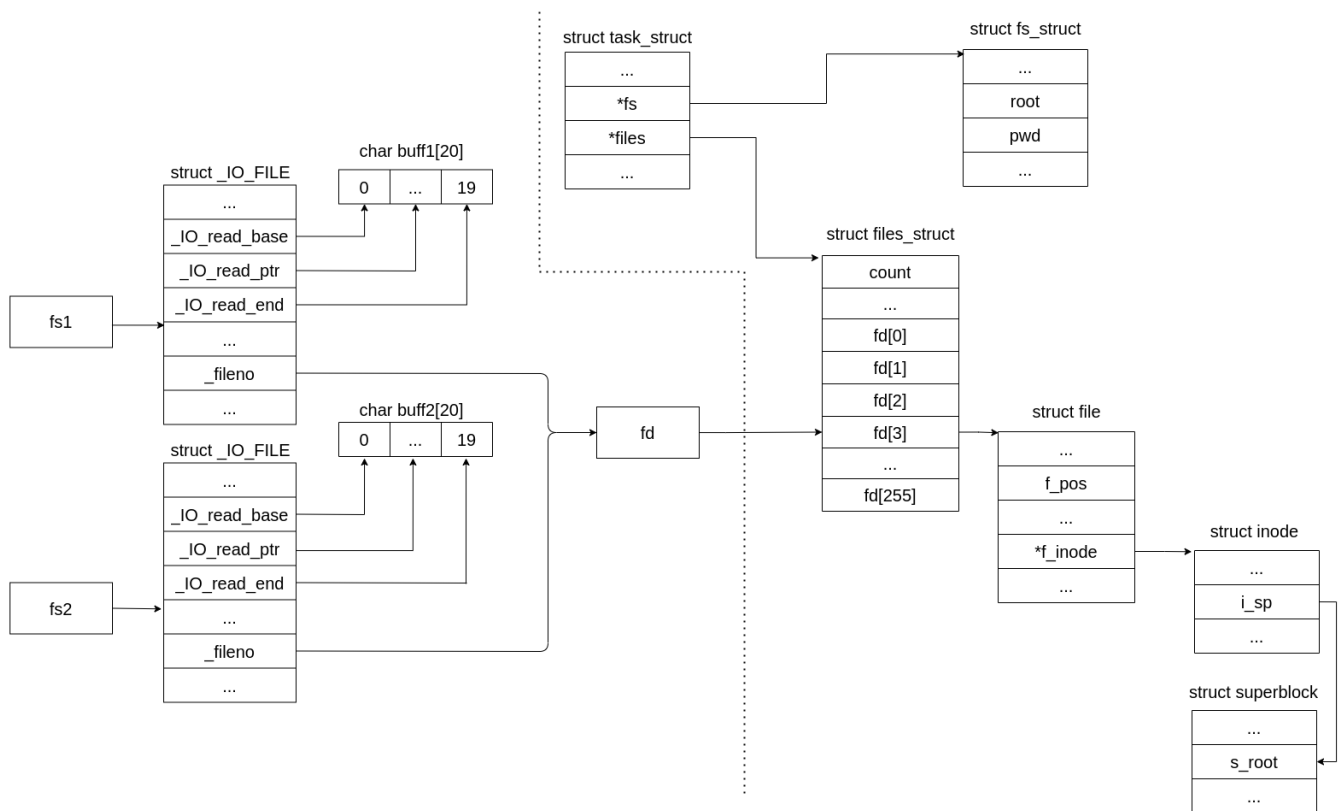


Рисунок 1.3 – Схема структур, используемых в первой программе

В начале работы программы системный вызов `open()` создает и возвращает файловый дескриптор для открытого только на чтение (флаг `O_RDONLY`) файла «data/alphabet.txt». Этот системный вызов также создает запись в общесистемной таблице открытых файлов, в которой указаны флаги и смещение в файле. Дескриптор файла является ссылкой на одну из этих записей. Данному файловому дескриптору присваивается значение 3, поскольку значения 0, 1, 2 зарезервированы потоками `stdin`, `stdout`, `stderr`.

В результате вызова `fdopen()` будут созданы 2 экземпляра структуры `FILE` (`fs1` и `fs2`), которые ссылаются на дескриптор, созданный ранее в результате системного вызова `open()`. Создаются буферы `buff1` и `buff2` размером 20 байт. Для дескрипторов `fs1` и `fs2` в результате вызова функции `setvbuf` устанавливаются буферы, а также тип буферизации `_OBF` (полная буферизация).

Затем происходит поочередный вызов функции `fscanf()` для `fs1` и `fs2`. В виду того, что установлена полная буферизация, при первом вызове `fscanf()` буфер `buff1` заполнится полностью, т.е. будет содержать 20 первых символов алфавита.

В результате вызовов функции `fscanf()` буфер `fs1` заполнится полностью, т.е. первыми 20 символами. Значение `f_pos` увеличится на 20.

В результате последующих вызовов функции `fscanf()` в буфер `fs2` считаются оставшиеся 6 символов из алфавита, начиная с `f_pos`.

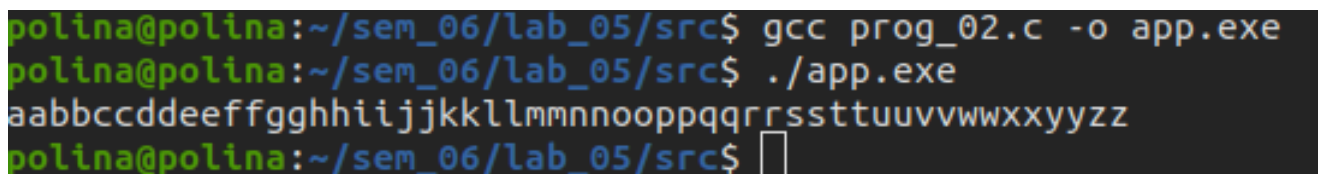
Затем в результате вызовов функции `fprintf()` поочередно выводятся символы из `buff1` и `buff2`.

1.5 ВТОРАЯ ПРОГРАММА

Листинг 1.5 – Программа №2 (один поток)

```
1 #include <fcntl.h>
2 #include <unistd.h>
3 #include <stdio.h>
4
5 #define OK 0
6 #define FILE_NAME "data/alphabet.txt"
7
8 int main(void)
9 {
10     int fd1 = open(FILE_NAME, O_RDONLY);
11     int fd2 = open(FILE_NAME, O_RDONLY);
12     int rc1, rc2 = 1;
13
14     char c;
15
16     while (rc1 == 1 || rc2 == 1)
17     {
18         if ((rc1 = read(fd1, &c, 1)) == 1)
19             write(1, &c, 1);
20
21         if ((rc2 = read(fd2, &c, 1)) == 1)
22             write(1, &c, 1);
23     }
24
25     fprintf(stdout, "\n");
26
27     return OK;
28 }
```

На рисунке 1.4 представлен результат работы второй программы.



```
polina@polina:~/sem_06/lab_05/src$ gcc prog_02.c -o app.exe
polina@polina:~/sem_06/lab_05/src$ ./app.exe
aabbccddeeffgghhiijjkkllmmnnooppqrrssttuuvvwxxyyzz
polina@polina:~/sem_06/lab_05/src$
```

Рисунок 1.4 – Результат работы второй программы (один поток)

Листинг 1.6 – Программа №2 (два потока)

```

1 #include <fcntl.h>
2 #include <stdio.h>
3 #include <unistd.h>
4 #include <pthread.h>
5
6 void *thread_run(void *fs)
7 {
8     int fd = *(int *)fs;
9     int rc = 1;
10
11     char c;
12     while (rc == 1)
13     {
14         rc = read(fd, &c, 1);
15         if (rc == 1)
16             write(1, &c, 1);
17     }
18
19     return NULL;
20 }
21
22 int main()
23 {
24     int fd1 = open("data/alphabet.txt", O_RDONLY);
25     int fd2 = open("data/alphabet.txt", O_RDONLY);
26     int rc = 1;
27
28     pthread_t td;
29     pthread_create(&td, NULL, thread_run, &fd2);
30     usleep(1);
31
32     char c;
33     while (rc == 1)
34     {
35         if ((rc = read(fd1, &c, 1)) == 1) write(1, &c, 1);
36     }
37
38     pthread_join(td, NULL);
39     fprintf(stdout, "\n");
40
41     return 0;
42 }

```

На рисунке 1.5 представлен результат работы второй программы (выполнено несколько запусков программы).

```

acegikmoqsuwypolina@polina:~/sem_06/lab_05/src$ gcc -pthread prog_02_thread.c -o app.exe
polina@polina:~/sem_06/lab_05/src$ ./app.exe
abcdefghijklmnopqrstuvwxyz
polina@polina:~/sem_06/lab_05/src$ ./app.exe
abcdefghijklmnopqrstuvwxyz
polina@polina:~/sem_06/lab_05/src$ ./app.exe
abcdefghijklmnopqrstuvwxyz
polina@polina:~/sem_06/lab_05/src$
```

Рисунок 1.5 – Результат работы второй программы (два потока)

1.6 АНАЛИЗ

На рисунке 1.6 представлена схема структур, используемых во второй программе.

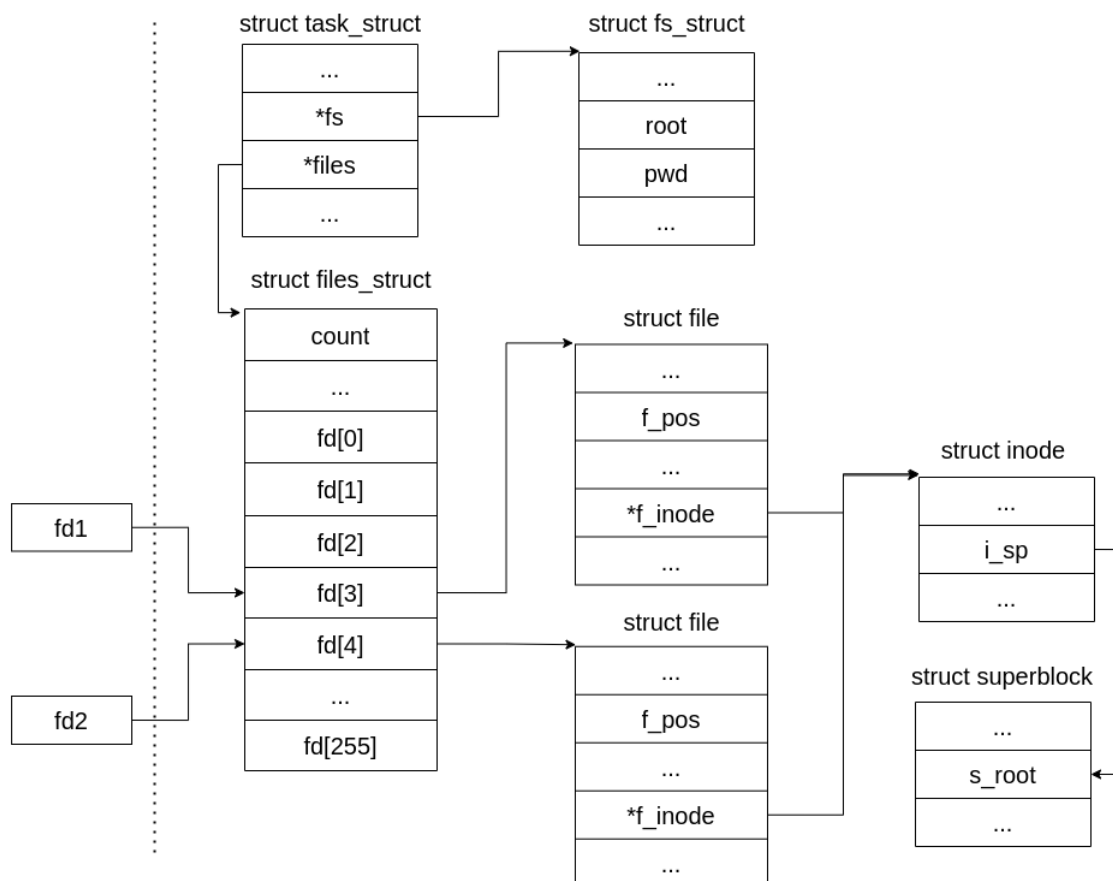


Рисунок 1.6 – Схема структур, используемых во второй программе

В начале работы программы в результате двукратного системного вызова `open()` создается 2 файловых дескриптора для одного и того же текстового файла, открытого только на чтение (`O_RDONLY`). Несмотря на то, что файл один и тот же, создаются две разные структуры `struct FILE`, ссылающиеся на один и тот же `struct inode`.

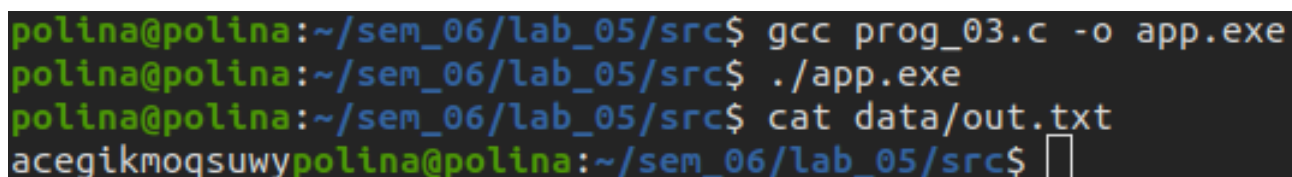
Т.к. созданы два экземпляра структуры, в случае однопоточной реализации посимвольная печать приведет к дублированию каждого символа при выводе на экран. В случае многопоточной реализации символы будут выводиться на экран в перемешанном порядке. Порядок символов вывода не гарантирован, однако алфавит будет выведен целиком.

1.7 ТРЕТЬЯ ПРОГРАММА

Листинг 1.7 – Программа №3 (один поток)

```
1 #include <fcntl.h>
2 #include <stdio.h>
3 #include <unistd.h>
4
5 int main()
6 {
7     FILE *fd1 = fopen("data/out.txt", "w");
8     FILE *fd2 = fopen("data/out.txt", "w");
9     for (char c = 'a'; c <= 'z'; ++c)
10    {
11        if (c % 2 == 0)
12            fprintf(fd1, "%c", c);
13        else
14            fprintf(fd2, "%c", c);
15    }
16    fclose(fd1);
17    fclose(fd2);
18    return 0;
19 }
```

На рисунке 1.7 представлен результат работы третьей программы.



```
polina@polina:~/sem_06/lab_05/src$ gcc prog_03.c -o app.exe
polina@polina:~/sem_06/lab_05/src$ ./app.exe
polina@polina:~/sem_06/lab_05/src$ cat data/out.txt
acegikmoqsuwpolina@polina:~/sem_06/lab_05/src$
```

Рисунок 1.7 – Результат работы третьей программы (один поток)

Листинг 1.8 – Программа №3 (два потока)

```
1 #include <fcntl.h>
2 #include <stdio.h>
3 #include <unistd.h>
4 #include <pthread.h>
```

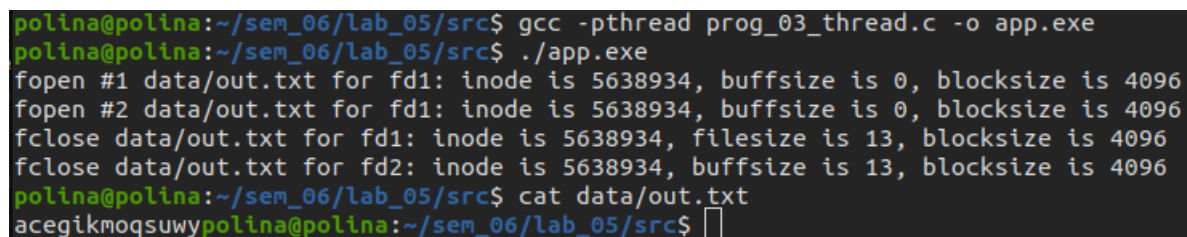
```

5 #include <sys/stat.h>
6
7 void *thread_run(void *fs)
8 {
9     struct stat info;
10
11     for (char c = 'a'; c <= 'z'; c += 2)
12     {
13         fprintf(fs, "%c", c);
14     }
15     fclose(fs);
16     stat("data/out.txt", &info);
17     fprintf(stdout, "fclose data/out.txt for fd2: inode is %ld, buffersize is %ld,
18         blocksize is %ld\n", info.st_ino, info.st_size, info.st_blksize);
19
20     return NULL;
21 }
22
23 int main()
24 {
25     struct stat info;
26
27     FILE *fd1 = fopen("data/out.txt", "w");
28     stat("data/out.txt", &info);
29     fprintf(stdout, "fopen #1 data/out.txt for fd1: inode is %ld, buffersize is %ld,
30         blocksize is %ld\n", info.st_ino, info.st_size, info.st_blksize);
31
32     FILE *fd2 = fopen("data/out.txt", "w");
33     stat("data/out.txt", &info);
34     fprintf(stdout, "fopen #2 data/out.txt for fd1: inode is %ld, buffersize is %ld,
35         blocksize is %ld\n", info.st_ino, info.st_size, info.st_blksize);
36
37     pthread_t td;
38     pthread_create(&td, NULL, thread_run, fd2);
39
40     for (char c = 'b'; c <= 'z'; c += 2)
41     {
42         fprintf(fd1, "%c", c);
43     }
44     fclose(fd1);
45     stat("data/out.txt", &info);
46     fprintf(stdout, "fclose data/out.txt for fd1: inode is %ld, filesize is %ld,
47         blocksize is %ld\n", info.st_ino, info.st_size, info.st_blksize);

```

```
44 | pthread_join(td, NULL);  
45 |  
46 | return 0;  
47 | }
```

На рисунке 1.8 представлен результат работы третьей программы.



```
polina@polina:~/sem_06/lab_05/src$ gcc -pthread prog_03_thread.c -o app.exe  
polina@polina:~/sem_06/lab_05/src$ ./app.exe  
fopen #1 data/out.txt for fd1: inode is 5638934, buffsize is 0, blocksize is 4096  
fopen #2 data/out.txt for fd1: inode is 5638934, buffsize is 0, blocksize is 4096  
fclose data/out.txt for fd1: inode is 5638934, filesize is 13, blocksize is 4096  
fclose data/out.txt for fd2: inode is 5638934, buffsize is 13, blocksize is 4096  
polina@polina:~/sem_06/lab_05/src$ cat data/out.txt  
acegikmoqsuwypolina@polina:~/sem_06/lab_05/src$
```

Рисунок 1.8 – Результат работы третьей программы (два потока)

1.8 АНАЛИЗ

На рисунке 1.9 представлена схема структур, используемых в третьей программе.

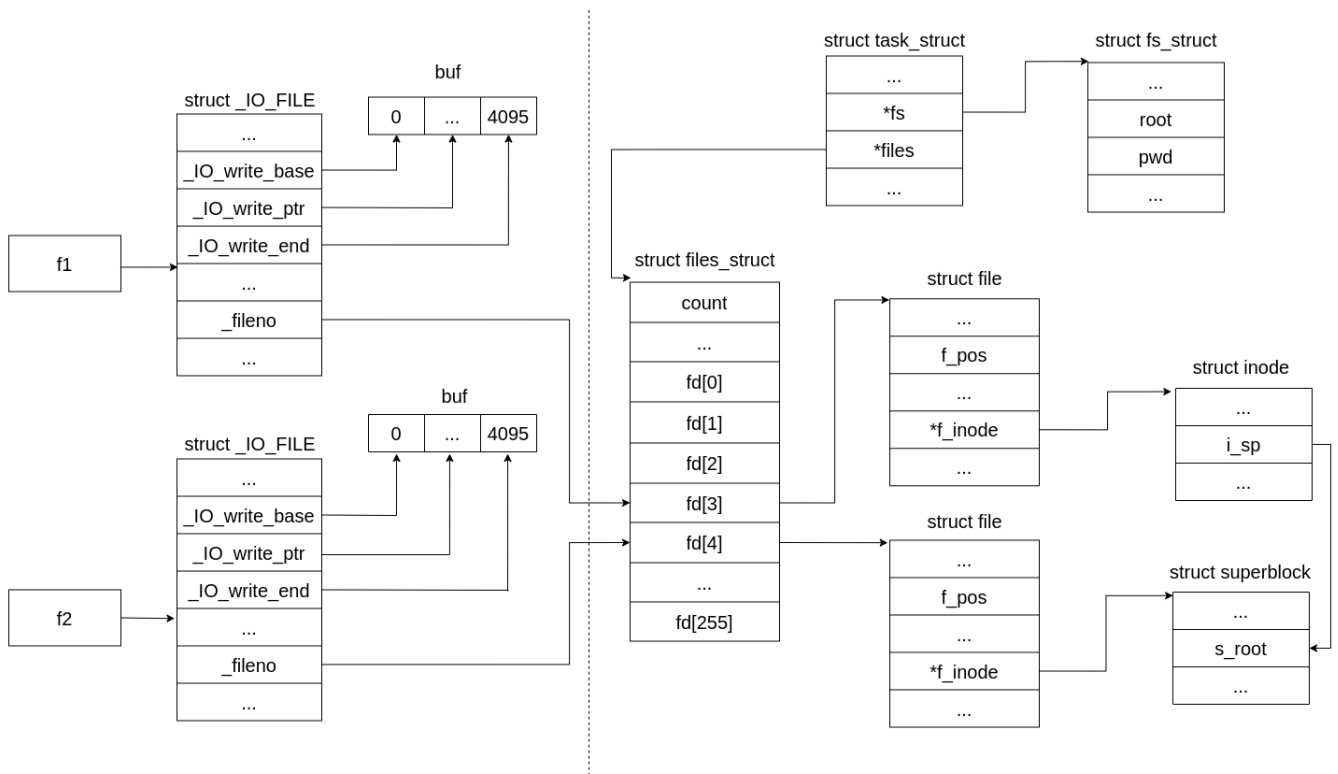


Рисунок 1.9 – Схема структур, используемых в третьей программе

В начале работы программы в результате двукратного вызова библиотечной функции `open()` файл открывается на запись 2 раза. Буфер создается без вмешательства программиста при первой операции записи, его размер равен 4096 байт (размер 1 страницы памяти). По умолчанию используется полная буферизация, при которой запись в файл из буфера произойдет либо при заполнении буфера, либо при вызове `fclose()`, либо при завершении процесса.

Вывод выполняется с помощью стандартной функции `fprintf()`

Информация будет записана в файл в следующих случаях:

1. Буфер полон;
2. Вызвана функция `fflush()` (Принудительная запись);
3. Если вызван `fclose()`.

В анализируемой программе информация из буфера запишется в файл в результате вызова `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()`.