



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                             */
34     char *_IO_save_end; /* Pointer to end of non-current get area. */
35
36     struct _IO_marker *_markers;
37
38     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

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    // read a char & write it alternatingly from fs1 and fs2
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
30        if ((flag2 = fscanf(fs2, "%c", &c)) == 1)
31        {
32            fprintf(stdout, "%c", c);
33        }
34    }
35    close(fd);
36
37    return 0;
38 }
```

```

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

```

Рис. 1.1: Результат работы первой программы

Листинг 1.2: Первая программа (реализация с потоками)

```

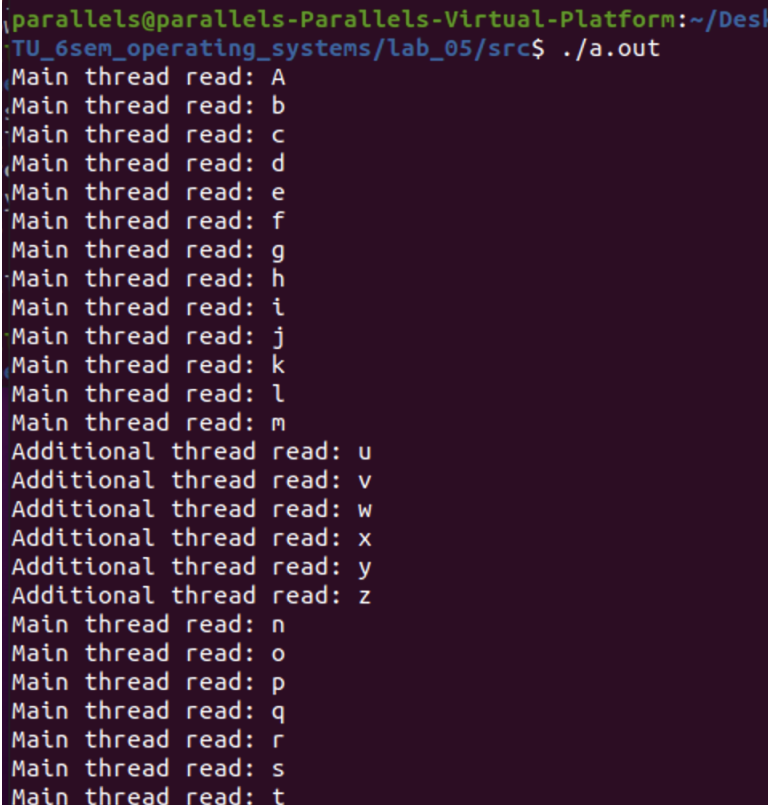
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <pthread.h>
4 #include <sys/types.h>
5 #include <sys/wait.h>
6 #include <unistd.h>
7
8 void *thread_func(void *args)
9 {
10     int flag = 1;
11     FILE *fs = (FILE *)args;
12
13     while (flag == 1)
14     {
15         char c;
16         if ((flag = fscanf(fs, "%c", &c)) == 1)
17         {
18             fprintf(stdout, "Additional thread read: %c\n", c);
19         }
20     }
21 }
22
23 int main(void)
24 {
25     setbuf(stdout, NULL);
26
27     pthread_t thread;
28     int fd = open("alphabet.txt", O_RDONLY);
29
30     FILE *fs1 = fdopen(fd, "r");
31     char buff1[20];
32     setvbuf(fs1, buff1, _IOFBF, 20);
33
34     FILE *fs2 = fdopen(fd, "r");
35     char buff2[20];
36     setvbuf(fs2, buff2, _IOFBF, 20);
37
38     if (pthread_create(&thread, NULL, thread_func, (void *)fs2) != 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 }
53 // The pthread_join() function suspends processing of the calling thread
54 // until the target thread completes.
55 // 2 arg – void **status
56 pthread_join(thread, NULL);
57
58 close(fd);
59
60 return 0;
61 }

```



```

parallels@parallels-Parallels-Virtual-Platform:~/Desk
TU_6sem_operating_systems/lab_05/src$ ./a.out
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
Additional thread read: u
Additional thread read: v
Additional thread read: w
Additional thread read: x
Additional thread read: y
Additional thread read: z
Main thread read: n
Main thread read: o
Main thread read: p
Main thread read: q
Main thread read: r
Main thread read: s
Main thread read: t

```

Рис. 1.2: Результат работы первой программы (с потоками)

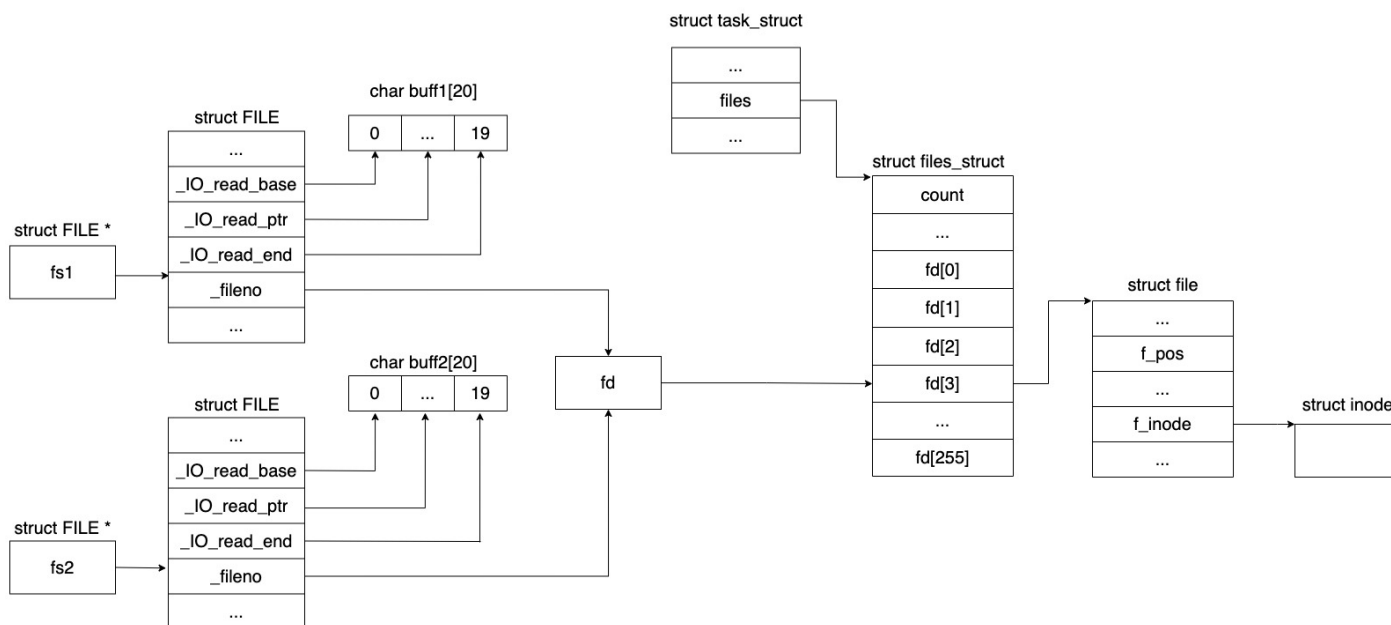


Рис. 1.3: Схема структур, используемых в первой программе

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 буфер и его размер (20 байт) и меняет тип буферизации на полную (`_IOFBF`).

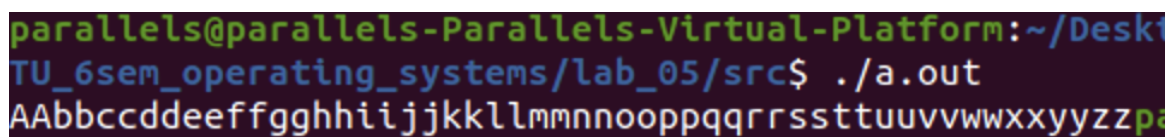
При первом вызове `fscanf()` (для `fs1`) буфер `buff1` полностью заполнится первыми 20 символами. Значение `f_pos` в структуре `struct_file` открытого файла увеличится на 20. При следующем вызове `fscanf()` (для `fs2`) в `buff2` считываются оставшиеся 6 символов, начиная с `f_pos` (`fs1` и `fs2` ссылаются 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 же (в данном запуске) главный поток начал в цикле выводить символы из своего буфера, затем потерял квант,??

## 2 | Вторая программа

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
19            rc2 = read(fd2, &c, 1);
20            if (rc2 == 1)
21            {
22                write(1, &c, 1);
23            }
24        }
25    }
26    close(fd1);
27    close(fd2);
28
29
30    return 0;
31 }
```



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

Рис. 2.1: 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
28     pthread_t thread;
29
30     if (pthread_create(&thread, NULL, thread_func, 0) != 0)
31     {
32         perror("error in pthread_create\n");
33         return -1;
34     }
35
36     int rc = 1;
37
38     while (rc == 1)
39     {
40         char c;
41         rc = read(fd1, &c, 1);
42         if (rc == 1)
43         {
44             write(1, &c, 1);
45         }
46     }
47
48     pthread_join(thread, NULL);
49     close(fd1);

```

```

50
51     return 0;
52 }

```

```

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

```

Рис. 2.2: Результат работы второй программы (с потоками)

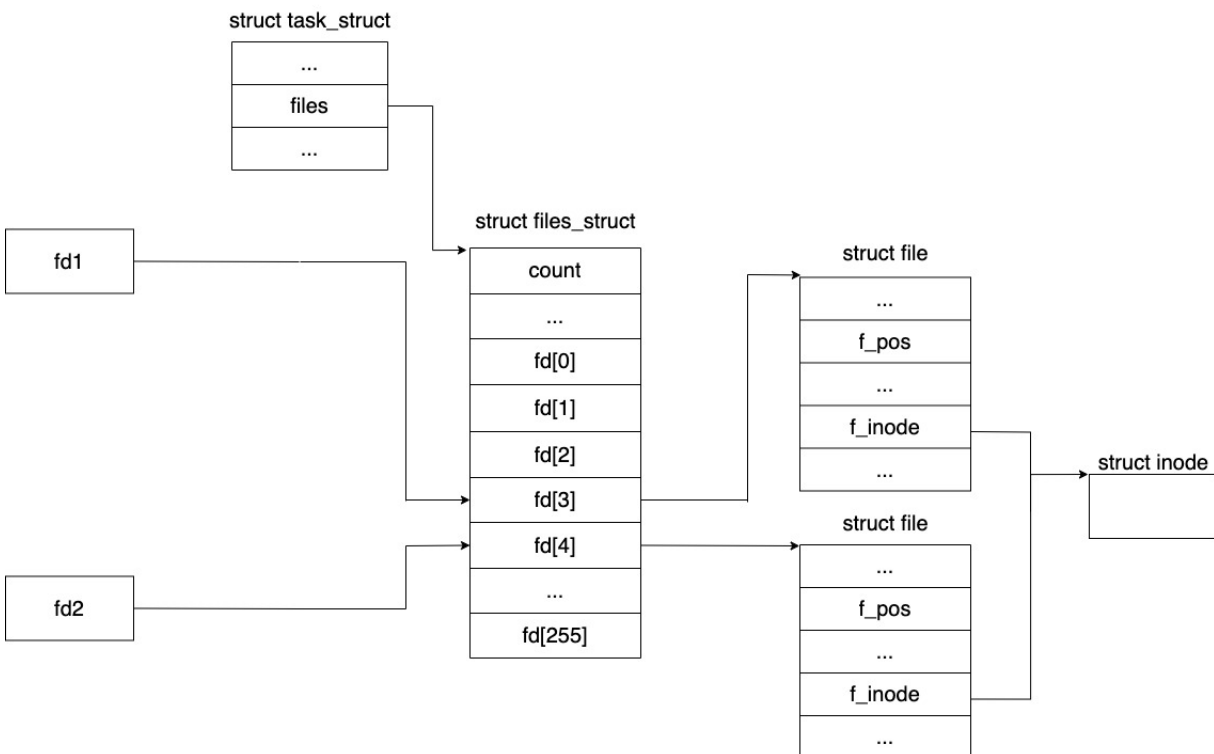


Рис. 2.3: Схема структур второй программы

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

На рис. 3.1 представлена схема структур, используемых в третьей программе.

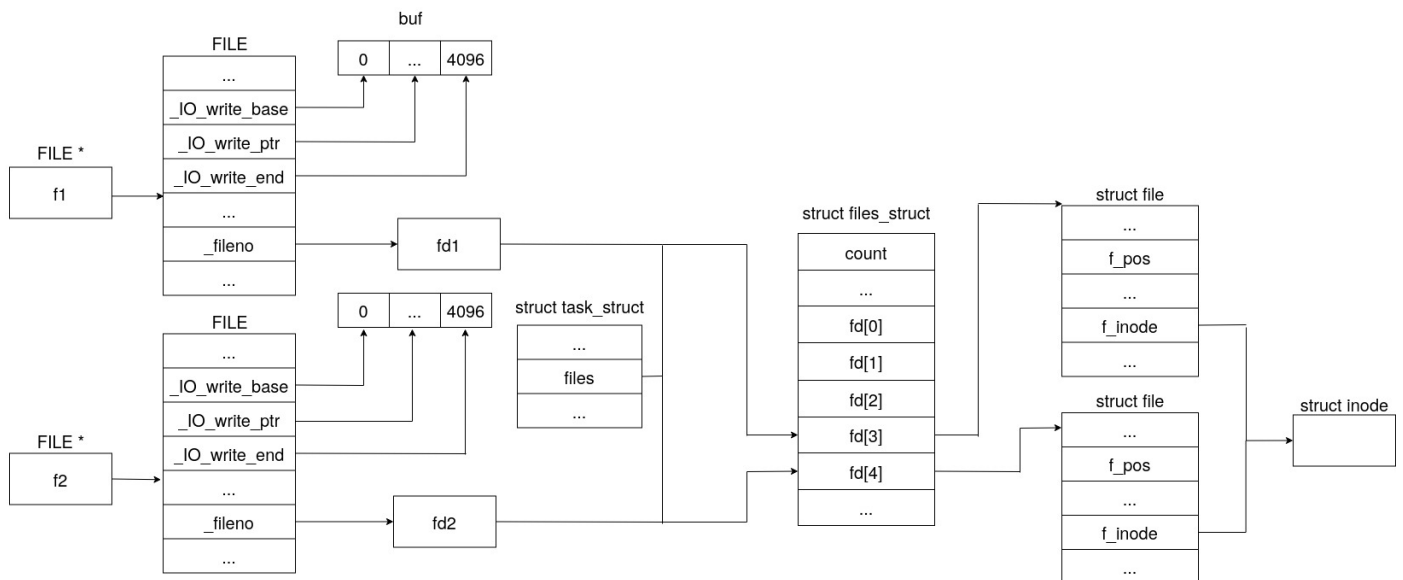


Рис. 3.1: Схема структур программы №2

- Файл открывается на запись два раза, с помощью функции `fopen()`;
- функция `fprintf()` предоставляет буферизованный вывод - буфер создается без нашего вмешательства;g
- изначально информация пишется в буфер, а из буфера в файл если произошло одно из событий:
  - буфер полон;
  - вызвана функция `fclose()`;
  - вызвана функция `fflush()`;
- в случае нашей программы, информация в файл запишется в результате вызова функция `fclose()`;
- из-за того `f_pos` независимы для каждого дескриптора файла, запись в файл будет производится с самого начала;

- таким образом, информация записанная при первом вызове `fclose()` будет потеряна в результате второго вызова `fclose()` (см. рис. 3.2).
- в многопоточной реализации результат аналогичен - с помощью `pthread_join` мы дождаемся вызова `fclose()` для `f2` в отдельном потоке и далее вызываем `fclose()` для `f1`.

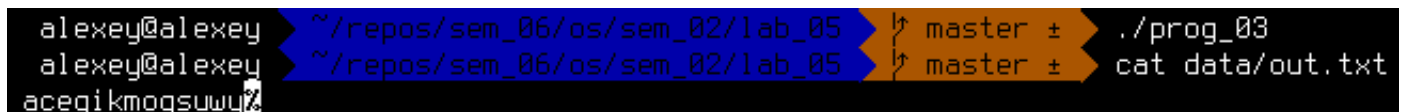
Листинг 3.1: Программа №3

```

1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #define OK 0
5 #define FILE_NAME "data/out.txt"
6 #define SPEC "%c"
7
8 int main()
9 {
10     FILE *f1 = fopen(FILE_NAME, "w");
11     FILE *f2 = fopen(FILE_NAME, "w");
12
13     for (char c = 'a'; c <= 'z'; c++)
14     {
15         if (c % 2)
16         {
17             fprintf(f1, SPEC, c);
18         }
19         else
20         {
21             fprintf(f2, SPEC, c);
22         }
23     }
24
25     fclose(f2);
26     fclose(f1);
27     return OK;
28 }

```

На рис. 3.2 представлен результат работы третьей программы.



```

alexey@alexey ~/repos/sem_06/os/sem_02/lab_05 ⌕ master ± ./prog_03
alexey@alexey ~/repos/sem_06/os/sem_02/lab_05 ⌕ master ± cat data/out.txt
acegikmoqsuwy

```

Рис. 3.2: Результат работы третьей программы

Листинг 3.2: Программа №3 (реализация с потоками)

```

1 #include <stdio.h>

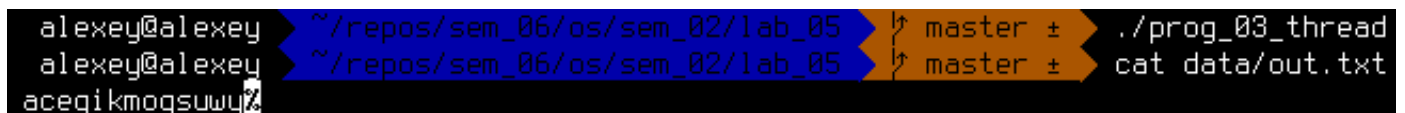
```

```

2 #include <fcntl.h>
3 #include <pthread.h>
4 #include <unistd.h>
5 #define OK 0
6 #define FILE_NAME "data/out.txt"
7 #define SPEC "%c"
8
9 void *run_buffer(void *args)
10 {
11     FILE *f = (FILE *)args;
12
13     for (char c = 'b'; c <= 'z'; c += 2)
14     {
15         fprintf(f, SPEC, c);
16     }
17
18     fclose(f);
19     return NULL;
20 }
21
22 int main()
23 {
24     FILE *f1 = fopen(FILE_NAME, "w");
25     FILE *f2 = fopen(FILE_NAME, "w");
26
27     pthread_t thread;
28     int rc = pthread_create(&thread, NULL, run_buffer, (void *)f2));
29
30     for (char c = 'a'; c <= 'z'; c += 2)
31     {
32         fprintf(f1, SPEC, c);
33     }
34
35     pthread_join(thread, NULL);
36     fclose(f1);
37     return OK;
38 }

```

На рис. 3.3 представлен результат работы второй программы (с потоками).



```

alexey@alexey ~/repos/sem_06/os/sem_02/lab_05 ⌕ master ± ./prog_03_thread
alexey@alexey ~/repos/sem_06/os/sem_02/lab_05 ⌕ master ± cat data/out.txt
acegikmoqsuwyz

```

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