



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 №5

Название: \_\_\_\_\_ Буферизованный и не буферизованный ввод-вывод

Дисциплина: \_\_\_\_\_ Операционные системы

Студент \_\_\_\_\_ ИУ7-64Б \_\_\_\_\_ С. Д. Параскун

Группа

Подпись, дата

И. О. Фамилия

Преподаватель \_\_\_\_\_ Н. Ю. Рязанова

Подпись, дата

И. О. Фамилия

Москва, 2022 г.

# 1. Программа №1

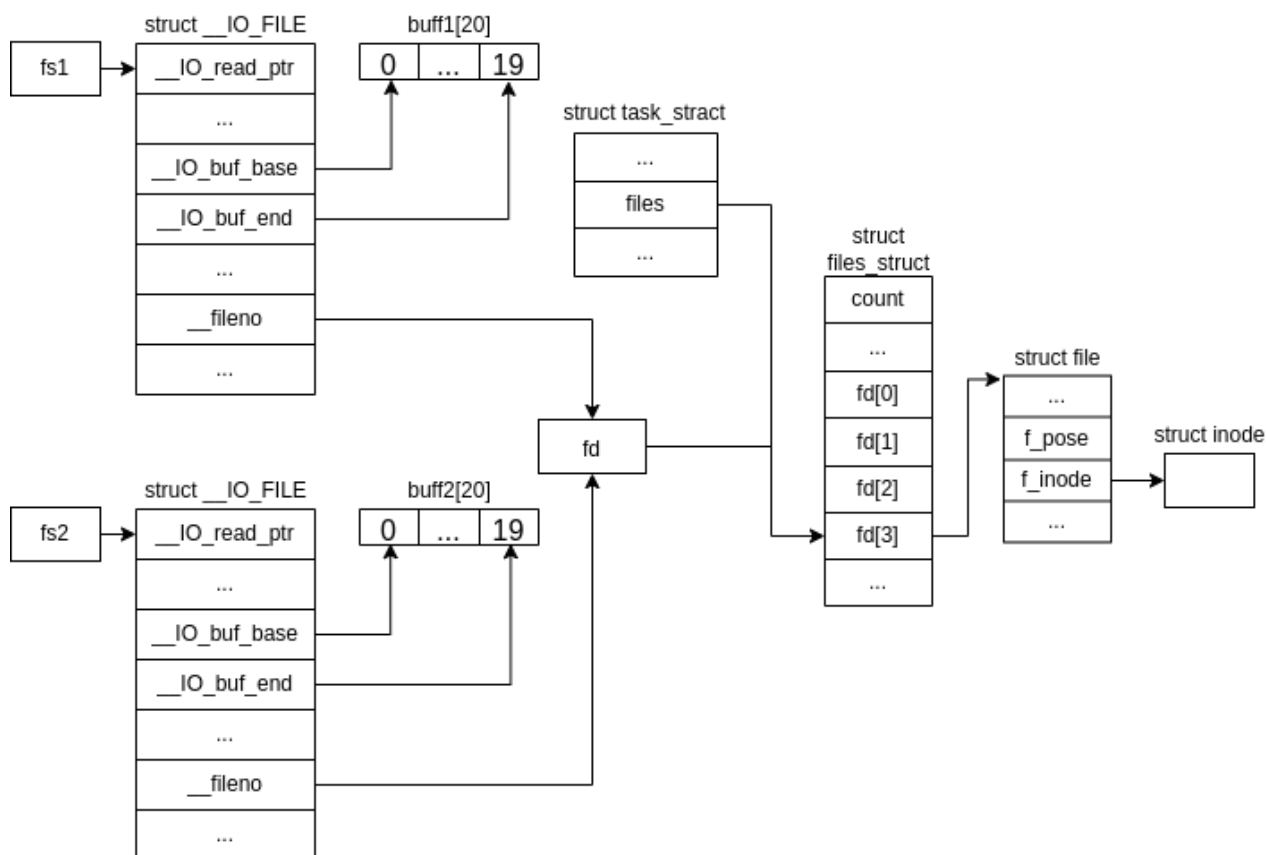


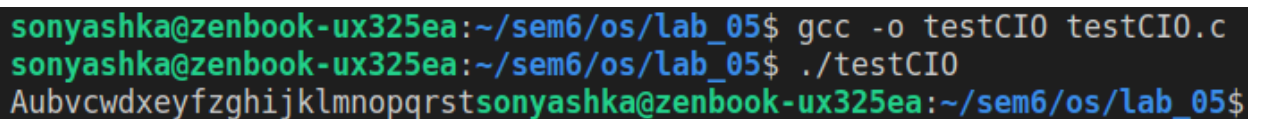
Рисунок 1.1 – Схема структур программы

Функция `open()` создает файловый дескриптор для файла `alphabet.txt` в системной таблице открытых файлов. Далее функция `fdopen()` создает экземпляр структуры `__IO_FILE` при каждом вызове, поле `__fileno` инициализируется значением дескриптора `fd`, возвращенного функцией `open()`. Функция `setvbuf()` явно задает размер буфера в 20 байт.

Первый `fscanf()` в цикле последовательно заполнит `buff1` первыми 20 символами алфавита, при этом значение поля `f_pos` в структуре `struct file` увеличится на 20. Второй вызов `fscanf()` в цикле последовательно заполнит `buff2` оставшимися 6 символами (начиная с `f_pos = 20`). Далее в цикле поочередно будут выводиться символы из `buff1` и `buff2`.

### Листинг 1.1 – Исходная программа

```
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            fprintf(stdout, "%c", c);
23        }
24        flag2 = fscanf(fs2, "%c", &c);
25        if (flag2 == 1) {
26            fprintf(stdout, "%c", c);
27        }
28    }
29    return 0;
30 }
```



```
sonyashka@zenbook-ux325ea:~/sem6/os/lab_05$ gcc -o testCI0 testCI0.c
sonyashka@zenbook-ux325ea:~/sem6/os/lab_05$ ./testCI0
Aubvcwdxeyfzghijklmnopqrstsonyashka@zenbook-ux325ea:~/sem6/os/lab_05$
```

### Рисунок 1.2 – Результат работы

В случае многопоточной реализации и механизма взаимного исключения дополнительный поток получит доступ к данным только тогда, когда главный поток прочитает файл до конца, т.е. `f_pos = 26`. Эту ситуацию можно разрешить создав два экземпляра дескрипторов открытых файлов.

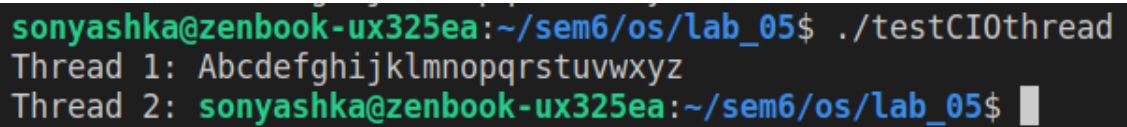
## Листинг 1.2 – Программа с дополнительным потоком

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <pthread.h>
4
5 typedef struct thread_args
6 {
7     FILE *f;
8     pthread_mutex_t *mutex;
9 } thread_args_t;
10
11 void run_thread(thread_args_t *args)
12 {
13     pthread_mutex_lock(args->mutex);
14     printf("\nThread 2: ");
15     int flag = 1;
16     while (flag == 1)
17     {
18         char c;
19         flag = fscanf(args->f, "%c", &c);
20         if (flag == 1)
21         {
22             fprintf(stdout, "%c", c);
23         }
24     }
25     pthread_mutex_unlock(args->mutex);
26 }
27
28 int main()
29 {
30     setbuf(stdout, NULL);
31     pthread_t thread;
32     int fd = open("alphabet.txt", O_RDONLY);
33
34     FILE *fs1 = fdopen(fd, "r");
35     char buff1[20];
36     setvbuf(fs1, buff1, _IOFBF, 20);
37
38     FILE *fs2 = fdopen(fd, "r");
39     char buff2[20];
40     setvbuf(fs2, buff2, _IOFBF, 20);
41
42     pthread_mutex_t mutex;
43     pthread_mutex_init(&mutex, NULL);
44
45     thread_args_t args;
46     args.f = fs2;
47     args.mutex = &mutex;
```

```

48 pthread_create(&thread, NULL, run_thread, &args);
49
50 pthread_mutex_lock(&mutex);
51 printf("Thread 1: ");
52 int flag = 1;
53 while(flag == 1)
54 {
55     char c;
56     flag = fscanf(fs1, "%c", &c);
57     if (flag == 1)
58     {
59         fprintf(stdout, "%c", c);
60     }
61 }
62 pthread_mutex_unlock(&mutex);
63 pthread_join(thread, NULL);
64 pthread_mutex_destroy(&mutex);
65 return 0;
66 }

```



```

sonyashka@zenbook-ux325ea:~/sem6/os/lab_05$ ./testCI0thread
Thread 1: Abcdefghijklmnopqrstuvwxyz
Thread 2: sonyashka@zenbook-ux325ea:~/sem6/os/lab_05$

```

Рисунок 1.3 – Результат работы

## 2. Программа №2

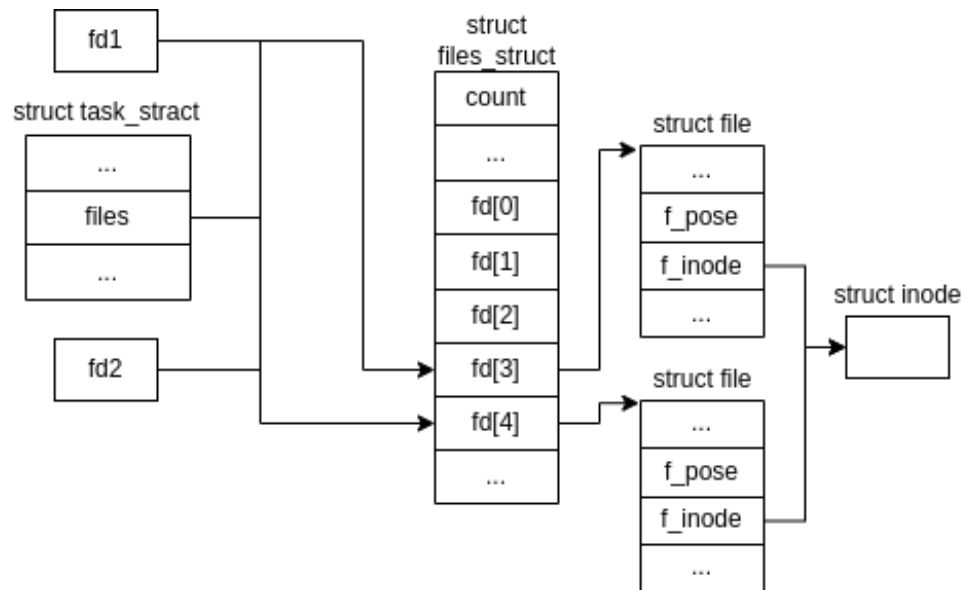


Рисунок 2.1 – Схема структур программы

Каждый вызов функции `open()` создает дескриптор открытого файла в системной таблице открытых файлов, ссылающихся на один `inode`. Для каждого экземпляра `struct file` значение поля `f_pos` меняется независимо, что позволяет избежать пропуска данных при чтении, используя разные дескрипторы.

Листинг 2.1 – Исходная программа

```
1 #include <fcntl.h>
2 int main()
3 {
4     char c;
5     int fd1 = open("alphabet.txt", O_RDONLY);
6     int fd2 = open("alphabet.txt", O_RDONLY);
7     int rc1 = 1, rc2 = 1;
8     while (rc1 == 1 && rc2 == 1)
9     {
10         if ((rc1 = read(fd1, &c, 1)) == 1)
11             write(1, &c, 1);
12         if ((rc2 = read(fd2, &c, 1)) == 1)
13             write(1, &c, 1);
14     }
15     return 0;
16 }
```

```
sonyashka@zenbook-ux325ea:~/sem6/os/lab_05$ ./testKernelIO
AAbbccddeeffgghhiijjkkllmmnnooppqrrssttuuvvwwxxyyzzsonyash
```

Рисунок 2.2 – Результат работы

Листинг 2.2 – Программа с дополнительным потоком

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <pthread.h>
4
5 typedef struct thread_args
6 {
7     int *fd;
8     pthread_mutex_t *mutex;
9 } thread_args_t;
10
11 void run_thread(thread_args_t *args)
12 {
13     pthread_mutex_lock(args->mutex);
14     char c;
15     int rc = 1;
16     while (rc == 1)
17     {
18         if ((rc = read(args->fd,&c,1)) == 1)
19             write(1,&c,1);
20     }
21     pthread_mutex_unlock(args->mutex);
22 }
23
24 int main()
25 {
26     char c;
27     pthread_t thread;
28     int fd1 = open("alphabet.txt",O_RDONLY);
29     int fd2 = open("alphabet.txt",O_RDONLY);
30     pthread_mutex_t mutex;
31     pthread_mutex_init(&mutex, NULL);
32
33     thread_args_t args;
34     args.fd = fd2;
35     args.mutex = &mutex;
36     pthread_create(&thread, NULL, run_thread, &args);
37     pthread_mutex_lock(&mutex);
38     int rc = 1;
39     while (rc == 1)
40     {
```

```
41         if ((rc = read(fd1,&c,1)) == 1)
42             write(1,&c,1);
43     }
44     pthread_mutex_unlock(&mutex);
45     pthread_join(thread, NULL);
46     pthread_mutex_destroy(&mutex);
47     return 0;
48 }
```

```
sonyashka@zenbook-ux325ea:~/sem6/os/lab_05$ ./testKernelIOthread
AbcdefghijklmnopqrstuvwxyzAbcdefghijklmnopqrstuvwxyzsonyashka@zen
```

Рисунок 2.3 – Результат работы



### 3. Программа №3

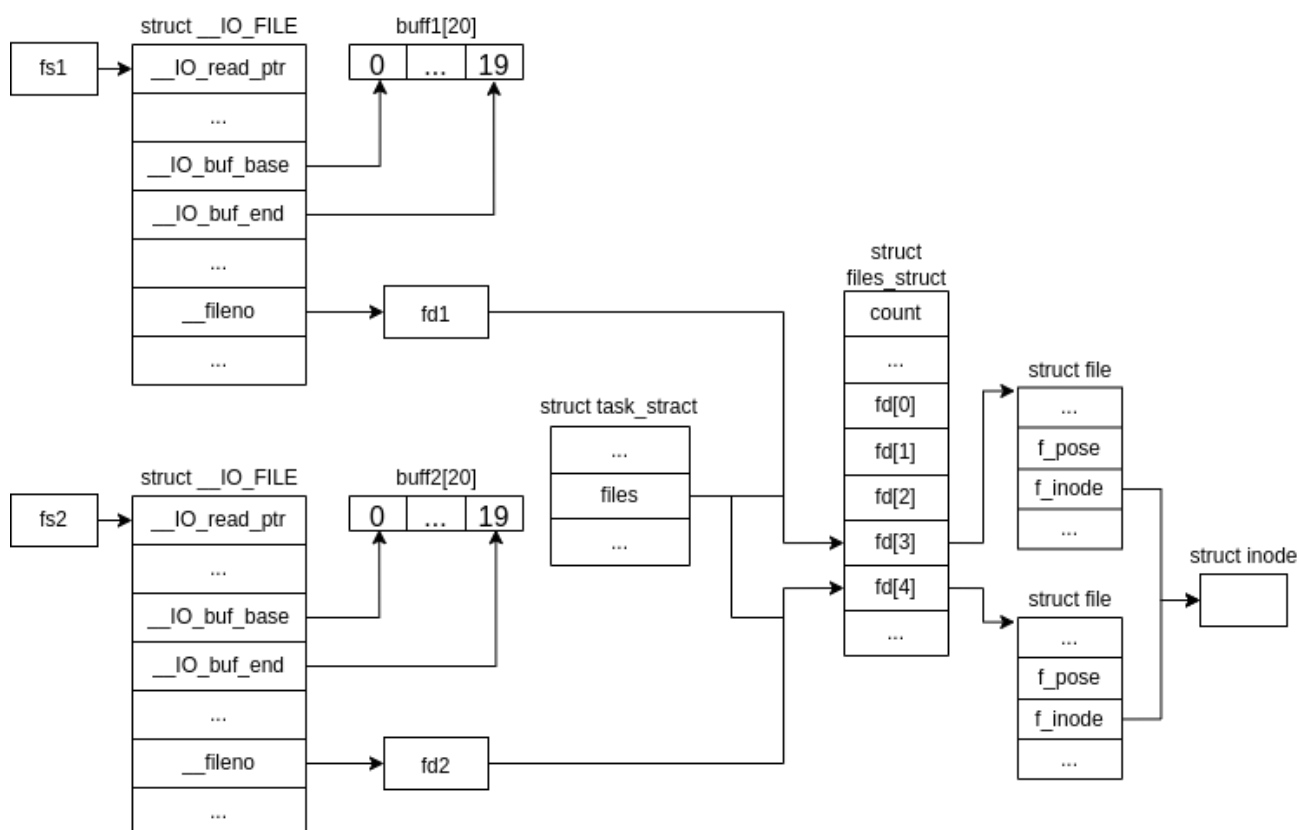


Рисунок 3.1 – Схема структур программы

Файл открывается два раза на запись с использованием функции `fopen()`, т.е. создается два экземпляра структуры `__IO_FILE`, каждому соответствует дескриптор открытого файла в системной таблице открытых файлов.

Функция `fprintf()` предоставляет буферизованный вывод. Информация будет записываться в буфер до тех пор, пока либо не переполнится буфер, либо не будут вызваны `fflush()` или `fclose()`. В случае программы ниже, запись произведется при вызове `fclose()`. Так как для каждого экземпляра созданы отдельные экземпляры `struct file`, то и значения полей `f_pos` будут изменяться независимо для каждого файла.

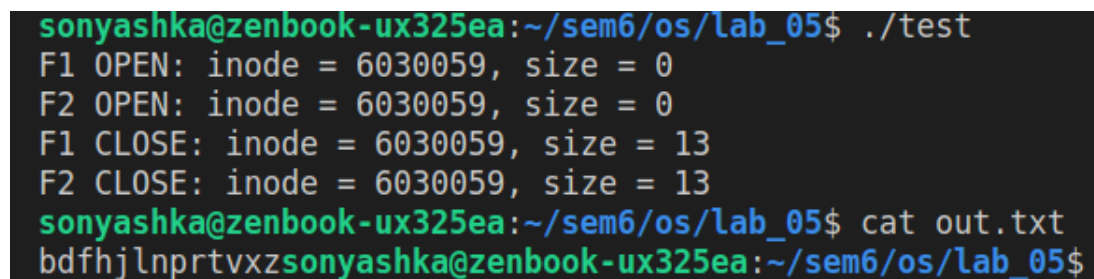
При первом вызове `fclose()` информация будет записана в файл. При следующем вызове `fclose()` данные записываются с начала файла, а предыдущая информация теряется. Данная проблема может быть решена использо-

ванием одного неделимого системного вызова O\_APPEND. В таком случае запись запись будет производиться в конец файла.

Также функция fopen() в качестве флагов принимает const char \*mode, в соответствии с которыми устанавливаются флаги для вызова open().

Листинг 3.1 – Исходная программа

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #define FILE_NAME "out.txt"
4
5 int main()
6 {
7     struct stat st1, st2;
8     FILE *f1 = fopen(FILE_NAME, "w");
9     stat(FILE_NAME, &st1);
10    printf("F1 OPEN: inode = %d, size = %d\n", st1.st_ino, st1.st_size);
11    FILE *f2 = fopen(FILE_NAME, "w");
12    stat(FILE_NAME, &st2);
13    printf("F2 OPEN: inode = %d, size = %d\n", st2.st_ino, st2.st_size);
14
15    for (char c = 'a'; c <= 'z'; c++)
16    {
17        if (c % 2)
18            fprintf(f1, "%c", c);
19        else
20            fprintf(f2, "%c", c);
21    }
22    fclose(f1);
23    stat(FILE_NAME, &st1);
24    printf("F1 CLOSE: inode = %d, size = %d\n", st1.st_ino, st1.st_size);
25    fclose(f2);
26    stat(FILE_NAME, &st2);
27    printf("F2 CLOSE: inode = %d, size = %d\n", st2.st_ino, st2.st_size);
28    return 0;
29 }
```



```
sonyashka@zenbook-ux325ea:~/sem6/os/lab_05$ ./test
F1 OPEN: inode = 6030059, size = 0
F2 OPEN: inode = 6030059, size = 0
F1 CLOSE: inode = 6030059, size = 13
F2 CLOSE: inode = 6030059, size = 13
sonyashka@zenbook-ux325ea:~/sem6/os/lab_05$ cat out.txt
bdfhjlnprtvxzs
sonyashka@zenbook-ux325ea:~/sem6/os/lab_05$
```

Рисунок 3.2 – Результат работы

## Листинг 3.2 – Программа с дополнительным потоком

```

1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <sys/stat.h>
4 #include <pthread.h>
5 #define FILE_NAME "outThread.txt"
6
7 void run_thread(FILE *f)
8 {
9     for (char c = 'b'; c <= 'z'; c += 2)
10     {
11         fprintf(f, "%c", c);
12     }
13 }
14
15 int main()
16 {
17     pthread_t thread;
18     struct stat st1, st2;
19     FILE *f1 = fopen(FILE_NAME, "w");
20     stat(FILE_NAME, &st1);
21     printf("F1 OPEN: inode = %d, size = %d\n", st1.st_ino, st1.st_size);
22     FILE *f2 = fopen(FILE_NAME, "w");
23     stat(FILE_NAME, &st2);
24     printf("F2 OPEN: inode = %d, size = %d\n", st2.st_ino, st2.st_size);
25
26     pthread_create(&thread, NULL, run_thread, f2);
27     for (char c = 'a'; c <= 'z'; c += 2)
28     {
29         fprintf(f1, "%c", c);
30     }
31
32     pthread_join(thread, NULL);
33     fclose(f1);
34     stat(FILE_NAME, &st1);
35     printf("F1 CLOSE: inode = %d, size = %d\n", st1.st_ino, st1.st_size);
36     fclose(f2);
37     stat(FILE_NAME, &st2);
38     printf("F2 CLOSE: inode = %d, size = %d\n", st2.st_ino, st2.st_size);
39     return 0;
40 }

```

```

sonyashka@zenbook-ux325ea:~/sem6/os/lab_05$ ./testthread
F1 OPEN: inode = 6030061, size = 0
F2 OPEN: inode = 6030061, size = 0
F1 CLOSE: inode = 6030061, size = 13
F2 CLOSE: inode = 6030061, size = 13
sonyashka@zenbook-ux325ea:~/sem6/os/lab_05$ cat outThread.txt
bdfhjlnprtvxzs
sonyashka@zenbook-ux325ea:~/sem6/os/lab_05$ █

```

Рисунок 3.3 – Результат работы

Ниже представлены листинги структур `struct stat` и `struct __IO_FILE`.

Листинг 3.3 – `struct stat`

```

1 struct stat {
2     dev_t      st_dev;      /* device */
3     ino_t      st_ino;      /* inode */
4     mode_t     st_mode;     /* access mode */
5     nlink_t    st_nlink;    /* number of hardlink */
6     uid_t      st_uid;      /* user id */
7     gid_t      st_gid;      /* group id */
8     dev_t      st_rdev;     /* device type */
9     /* (if its a device) */
10    off_t       st_size;     /* size in bytes */
11    blksize_t   st_blksize;  /* size of IO-block */
12    /* in FS */
13    blkcnt_t    st_blocks;   /* count of allocated blocks */
14    time_t      st_atime;    /* time of last access */
15    time_t      st_mtime;    /* time of last modification */
16    time_t      st_ctime;    /* time of last changing */
17 };

```

Листинг 3.4 – `struct __IO_FILE`

```

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

```

```

15     char *_IO_backup_base; /*Pointer to firstvalid character of backuparea*/
16     char *_IO_save_end; /* Pointer to end of non-current get area. */
17     struct _IO_marker *_markers;
18     struct _IO_FILE *_chain;
19     int _fileno;
20     int _flags2;
21     __off_t _old_offset; /* This used to be _offset but it's too small. */
22     /* 1+column number of pbase(); 0 is unknown. */
23     unsigned short _cur_column;
24     signed char _vtable_offset;
25     char _shortbuf[1];
26     _IO_lock_t *_lock;
27     #ifdef _IO_USE_OLD_IO_FILE
28 };

```