



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 №10
по курсу «Операционные системы»
на тему: «Буферизованный и не буферизованный ввод-вывод»

Студент	<u>ИУ7-63Б</u>	_____	<u>Лагутин Д. В.</u>
	(Группа)	(Подпись, дата)	(Фамилия И. О.)
Преподаватель		_____	<u>Рязанова Н. Ю.</u>
		(Подпись, дата)	(Фамилия И. О.)

Москва, 2023 г.

1 Используемые структуры

— Версия ядра: 6.4

— Версия glibc: 2.34

Листинг 1 – struct _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
18                           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 };
37
38 struct _IO_FILE_complete
39 {
```

```

40     struct _IO_FILE _file;
41 #endif
42     __off64_t _offset;
43     /* Wide character stream stuff. */
44     struct _IO_codecvt *_codecvt;
45     struct _IO_wide_data *_wide_data;
46     struct _IO_FILE *_freeres_list;
47     void *_freeres_buf;
48     size_t __pad5;
49     int _mode;
50     /* Make sure we don't get into trouble again. */
51     char _unused2[15 * sizeof (int) - 4 * sizeof (void *)
52                 - sizeof (size_t)];
53 };

```

Листинг 2 – struct file

```

1 struct file {
2     union {
3         struct llist_node      f_llist;
4         struct rcu_head        f_rcuhead;
5         unsigned int           f_iocb_flags;
6     };
7     struct path                f_path;
8     struct inode               *f_inode;      /* cached value */
9     const struct file_operations *f_op;
10
11     /*
12      * Protects f_ep, f_flags.
13      * Must not be taken from IRQ context.
14      */
15     spinlock_t                f_lock;
16     atomic_long_t              f_count;
17     unsigned int               f_flags;
18     fmode_t                    f_mode;
19     struct mutex                f_pos_lock;
20     loff_t                     f_pos;
21     struct fown_struct          f_owner;
22     const struct cred           *f_cred;
23     struct file_ra_state       f_ra;
24
25     u64                         f_version;
26 #ifdef CONFIG_SECURITY
27     void                        *f_security;
28 #endif
29     /* needed for tty driver, and maybe others */
30     void                        *private_data;
31
32 #ifdef CONFIG_EPOLL

```

```

33      /* Used by fs/eventpoll.c to link all the hooks to this file */
34      struct hlist_head      *f_ep;
35 #endif /* #ifdef CONFIG_EPOLL */
36      struct address_space    *f_mapping;
37      errseq_t                f_wb_err;
38      errseq_t                f_sb_err; /* for syncfs */
39 };

```

Листинг 3 – struct path

```

1 struct path {
2     struct vfsmount *mnt;
3     struct dentry *dentry;
4 };

```

Листинг 4 – struct stat

```

1 struct stat
2 {
3     /* These are the members that POSIX.1 requires.  */
4
5     __mode_t st_mode;                /* File mode.  */
6 #ifndef __USE_FILE_OFFSET64
7     __ino_t st_ino;                  /* File serial number.  */
8 #else
9     __ino64_t st_ino;                /* File serial number.  */
10 #endif
11     __dev_t st_dev;                  /* Device containing the file.  */
12     __nlink_t st_nlink;              /* Link count.  */
13
14     __uid_t st_uid;                  /* User ID of the file's owner.  */
15     __gid_t st_gid;                  /* Group ID of the file's group.  */
16 #ifndef __USE_FILE_OFFSET64
17     __off_t st_size;                 /* Size of file, in bytes.  */
18 #else
19     __off64_t st_size;               /* Size of file, in bytes.  */
20 #endif
21
22     __time_t st_atime;               /* Time of last access.  */
23     __time_t st_mtime;               /* Time of last modification.  */
24     __time_t st_ctime;               /* Time of last status change.  */
25
26     /* This should be defined if there is a 'st_blksize' member.  */
27 #undef  _STATBUF_ST_BLKSIZE
28 };

```

2 Программа 1

2.1 Однопоточная программа

Листинг 5 – Первая программа (однопоточный вариант)

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <fcntl.h>
4 #include <unistd.h>
5
6 #define MAX_FDS 10
7 #define MAX_BUF_SIZE 100
8
9 struct arg
10 {
11     int valid;
12     size_t amount;
13     size_t buf_size;
14 };
15
16 int main(int argc, char **argv)
17 {
18     const struct arg arg = parse_args(argc, argv);
19
20     if (EXIT_SUCCESS != arg.valid)
21         return EXIT_FAILURE;
22
23     FILE *fds[MAX_FDS] = {NULL};
24     char buf[MAX_FDS][MAX_BUF_SIZE], c;
25     int fd = open("alphabet", O_RDONLY), rc = EXIT_SUCCESS;
26
27     if (0 > fd)
28     {
29         perror("open error\n");
30         rc = EXIT_FAILURE;
31     }
32
33     for (size_t i = 0; EXIT_SUCCESS == rc && arg.amount > i; i++)
34     {
35         fds[i] = fdopen(fd, "r");
36
37         if (!fds[i])
38         {
```

```

39         perror("fdopen error\n");
40         rc = EXIT_FAILURE;
41     }
42     else if (EXIT_SUCCESS
43             != setvbuf(fds[i], buf[i], _IOFBF, arg.buf_size))
44     {
45         perror("setvbuf error\n");
46         rc = EXIT_FAILURE;
47     }
48 }
49
50 if (EXIT_SUCCESS == rc)
51     for (int r = 1; r && !(r = 0);)
52         for (size_t i = 0; arg.amount > i; i++)
53             if (1 == fscanf(fds[i], "%c", &c))
54             {
55                 fprintf(stdout, "%c\n", c);
56                 r = 1;
57             }
58
59 for (size_t i = 0; arg.amount > i; i++)
60     if (fds[i])
61         fclose(fds[i]);
62
63 if (0 <= fd)
64     close(fd);
65
66 return rc;
67 }

```

```
$ ./out/task1.out
a
u
b
v
c
w
d
x
e
y
f
z
g
h
i
j
k
l
m
n
o
p
q
r
s
t
```

Рисунок 2.1 – Результат работы первой программы

В результате работы описанной выше программы можно видеть, что при работе со вторым файловым дескриптором вывод начинается с символа «u» (20 в алфавите), что происходит из-за наличия буфера. При первом вызове `fscanf` в буфер первого файлового дескриптора считываются 20 символов, второго — оставшиеся 6.

2.2 Многопоточная программа

Листинг 6 – Первая программа (многопоточный вариант)

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <fcntl.h>
4 #include <unistd.h>
5 #include <pthread.h>
6
```

```

7 | #define MAX_THREADS 12
8 | #define MAX_BUF_SIZE 200
9 |
10 | struct data
11 | {
12 |     int valid;
13 |     size_t threads;
14 |     size_t buf_size;
15 |     int fd;
16 | };
17 |
18 | struct handle
19 | {
20 |     size_t id;
21 |     const struct data *data;
22 | };
23 |
24 | void *thread_func(void *_arg)
25 | {
26 |     struct handle *arg = _arg;
27 |     char buf[MAX_BUF_SIZE], tmp;
28 |     int rc = EXIT_SUCCESS;
29 |
30 |     FILE *f = fdopen(arg->data->fd, "r");
31 |
32 |     if (!f)
33 |     {
34 |         perror("fdopen error\n");
35 |         rc = EXIT_FAILURE;
36 |     }
37 |     else if (EXIT_SUCCESS != setvbuf(f, buf, _IOFBF, arg->data->buf_size))
38 |     {
39 |         perror("setvbuf error\n");
40 |         rc = EXIT_FAILURE;
41 |     }
42 |
43 |     if (EXIT_SUCCESS == rc)
44 |         for (int read = 1; read;)
45 |             if ((read = (1 == fscanf(f, "%c", &tmp))))
46 |                 fprintf(stdout, "thread %zu: %c\n", arg->id, tmp);
47 |
48 |     if (f)
49 |         fclose(f);
50 |
51 |     return NULL;
52 | }
53 |
54 | int main(int argc, char **argv)

```



```

55 {
56     struct data data = parse_args(argc, argv);
57
58     if (EXIT_SUCCESS != data.valid)
59         return EXIT_FAILURE;
60
61     struct handle args[MAX_THREADS];
62     pthread_t threads[MAX_THREADS];
63
64     int rc = EXIT_SUCCESS;
65     data.fd = open("alphabet", O_RDONLY);
66
67     if (0 > data.fd)
68     {
69         perror("open error\n");
70         rc = EXIT_FAILURE;
71     }
72
73     for (size_t i = 0; EXIT_SUCCESS == rc && data.threads > i; i++)
74     {
75         args[i].id = i + 1;
76         args[i].data = &data;
77
78         if (EXIT_SUCCESS
79             != pthread_create(threads + i, NULL, thread_func, args + i))
80         {
81             perror("pthread_create error\n");
82             rc = EXIT_FAILURE;
83         }
84     }
85
86     for (size_t i = 0; data.threads > i; i++)
87         pthread_join(threads[i], NULL);
88
89     if (0 <= data.fd)
90         close(data.fd);
91
92     return rc;
93 }

```

```
$ ./out/task1p.out
thread 2: a
thread 2: b
thread 2: c
thread 2: d
thread 2: e
thread 2: f
thread 2: g
thread 2: h
thread 2: i
thread 2: j
thread 2: k
thread 2: l
thread 2: m
thread 2: n
thread 2: o
thread 2: p
thread 2: q
thread 2: r
thread 2: s
thread 2: t
thread 1: u
thread 1: v
thread 1: w
thread 1: x
thread 1: y
thread 1: z
```

Потоки: 2, Буфер: 20

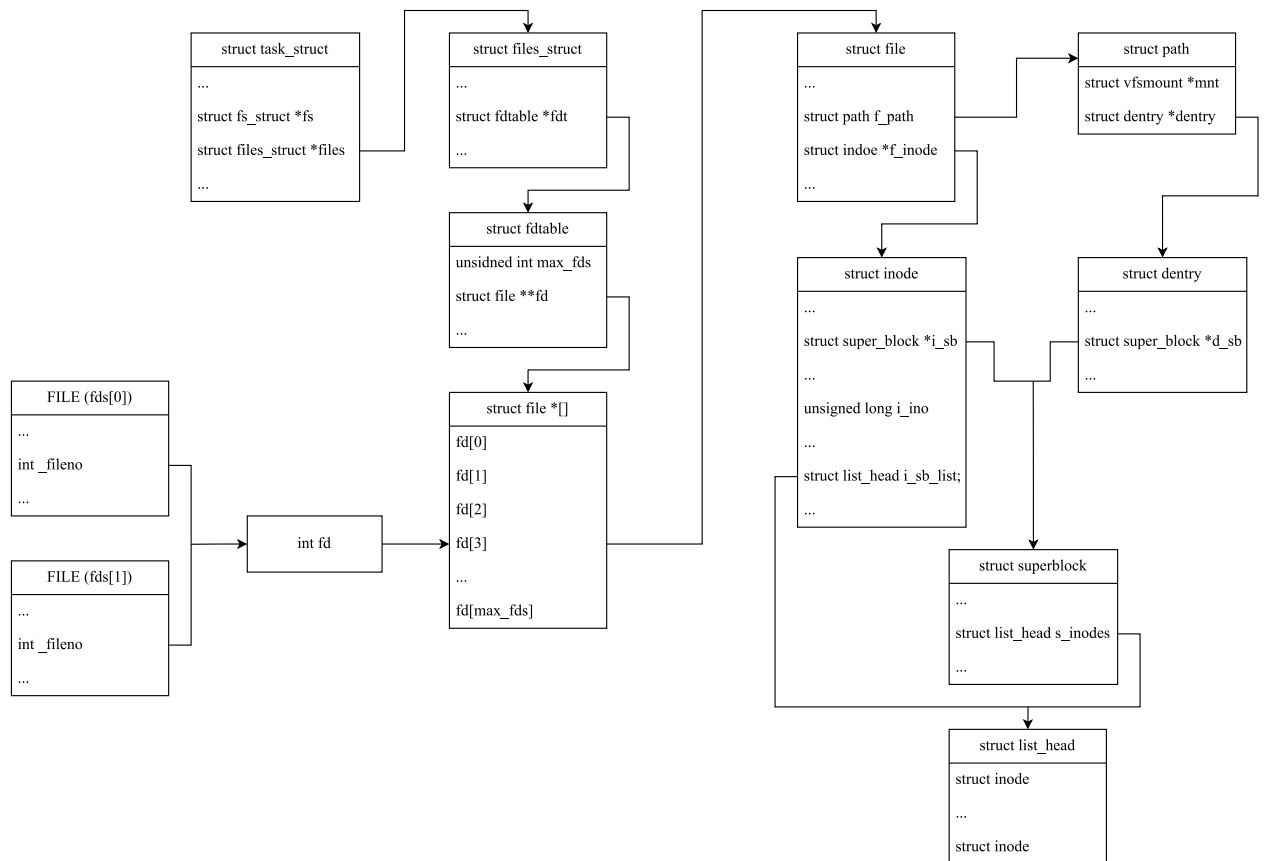
```
$ ./out/task1p.out 3 5
thread 2: a
thread 2: b
thread 2: c
thread 2: d
thread 2: e
thread 2: k
thread 2: l
thread 2: m
thread 2: n
thread 2: o
thread 1: f
thread 1: g
thread 1: h
thread 1: i
thread 1: j
thread 2: p
thread 2: q
thread 2: r
thread 2: s
thread 3: u
thread 3: v
thread 3: w
thread 3: x
thread 2: t
thread 3: y
thread 1: z
```

Потоки: 3, Буфер: 5

Рисунок 2.2 – Результат работы первой программы.

В многопоточном варианте программы чтение также буферизовано. Все потоки считывают из файла количество байт, равное размеру буфера, а так как работа потоков не синхронизирована, то и последовательность вывода нарушается.

2.3 Связь структур



3 Программа 2

3.1 Однопоточная программа

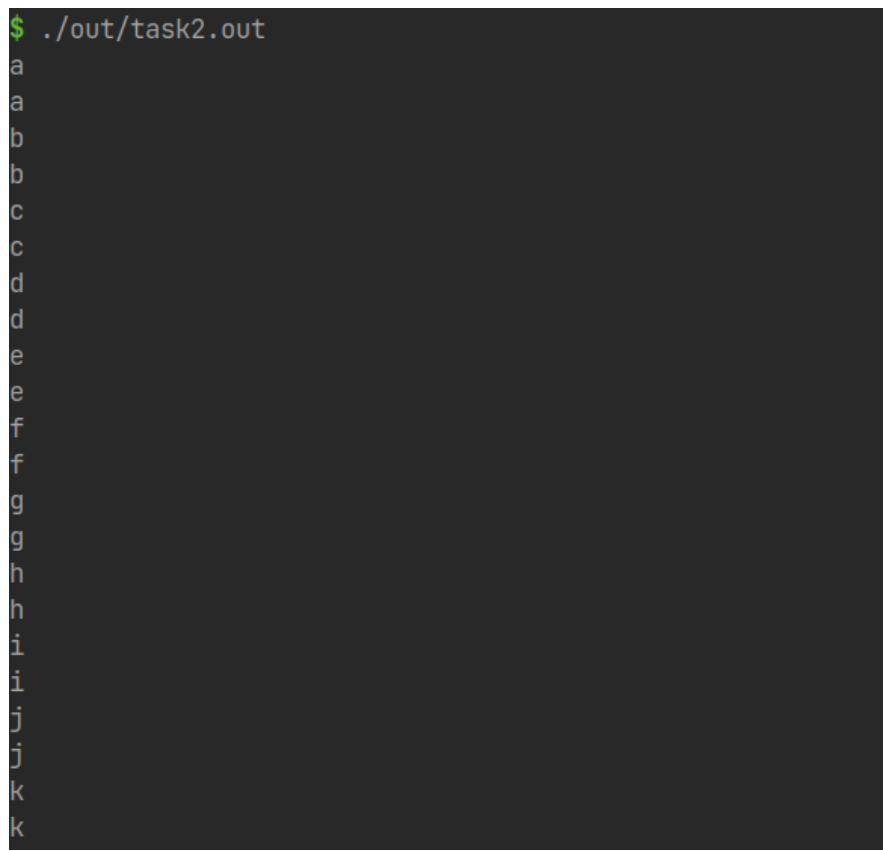
Листинг 7 – Вторая программа (однопоточный вариант)

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <fcntl.h>
4 #include <unistd.h>
5
6 #define MAX_FDS 10
7
8 struct arg
9 {
10     int valid;
11     size_t amount;
12 };
13
14 int main(int argc, char **argv)
15 {
16     const struct arg arg = parse_args(argc, argv);
17
18     if (EXIT_SUCCESS != arg.valid)
19         return EXIT_FAILURE;
20
21     int fds[MAX_FDS] = {-1};
22     char buf[2] = "\0\n";
23     int rc = EXIT_SUCCESS;
24
25     for (size_t i = 0; EXIT_SUCCESS == rc && arg.amount > i; i++)
26     {
27         fds[i] = open("alphabet", O_RDONLY);
28
29         if (0 > fds[i])
30         {
31             perror("open error");
32             rc = EXIT_FAILURE;
33         }
34     }
35
36     if (EXIT_SUCCESS == rc)
37         for (int r = 1; r && !(r = 0);)
38             for (size_t i = 0; arg.amount > i; i++)
```

```

39         if (1 == read(fds[i], buf, 1))
40         {
41             write(1, buf, 2);
42             r = 1;
43         }
44
45     for (size_t i = 0; arg.amount > i; i++)
46         if (fds[i] >= 0)
47             close(fds[i]);
48
49     return rc;
50 }

```



```

$ ./out/task2.out
a
a
b
b
c
c
d
d
e
e
f
f
g
g
h
h
i
i
j
j
k
k

```

Рисунок 3.1 – Результат работы второй программы

В данной программе процесс получает несколько дескрипторов открытого файла с использованием системного вызова `open`. В результате этого чтение происходит независимо, что проявляется в дублировании выводимых символов.

3.2 Многопоточная программа

Листинг 8 – Вторая программа (многопоточный вариант)

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <fcntl.h>
4 #include <unistd.h>
5 #include <pthread.h>
6
7 #define MAX_THREADS 12
8
9 struct arg
10 {
11     int valid;
12     size_t threads;
13 };
14
15 void *thread_func(void *_arg)
16 {
17     int fd = open("alphabet", O_RDONLY);
18     char tmp, buf[30];
19     ssize_t len;
20
21     if (0 > fd)
22     {
23         perror("open error\n");
24
25         return NULL;
26     }
27
28     for (int read1 = 1; read1;)
29         if ((read1 = (1 == read(fd, &tmp, 1))))
30         {
31             len = sprintf(buf, "thread %zu: %c\n", *(size_t *)_arg, tmp);
32             write(1, buf, len);
33         }
34
35     close(fd);
36
37     return NULL;
38 }
39
40 int main(int argc, char **argv)
41 {
42     const struct arg arg = parse_args(argc, argv);
```

```

43
44     if (EXIT_SUCCESS != arg.valid)
45         return EXIT_FAILURE;
46
47     size_t args[MAX_THREADS];
48     pthread_t threads[MAX_THREADS];
49     int rc = EXIT_SUCCESS;
50
51     for (size_t i = 0; EXIT_SUCCESS == rc && arg.threads > i; i++)
52     {
53         args[i] = i + 1;
54
55         if (EXIT_SUCCESS
56             != pthread_create(threads + i, NULL, thread_func, args + i))
57         {
58             perror("pthread_create error\n");
59             rc = EXIT_FAILURE;
60         }
61     }
62
63     for (size_t i = 0; arg.threads > i; i++)
64         pthread_join(threads[i], NULL);
65
66     return rc;
67 }

```

```
$ ./out/task2p.out
```

```

thread 1: a
thread 2: a
thread 1: b
thread 2: b
thread 1: c
thread 2: c
thread 1: d
thread 2: d
thread 1: e
thread 2: e
thread 1: f
thread 2: f
thread 1: g
thread 2: g
thread 1: h
thread 2: h
thread 1: i
thread 2: i
thread 1: j
thread 2: j
thread 1: k
thread 2: k

```

```
$ ./out/task2p.out
```

```

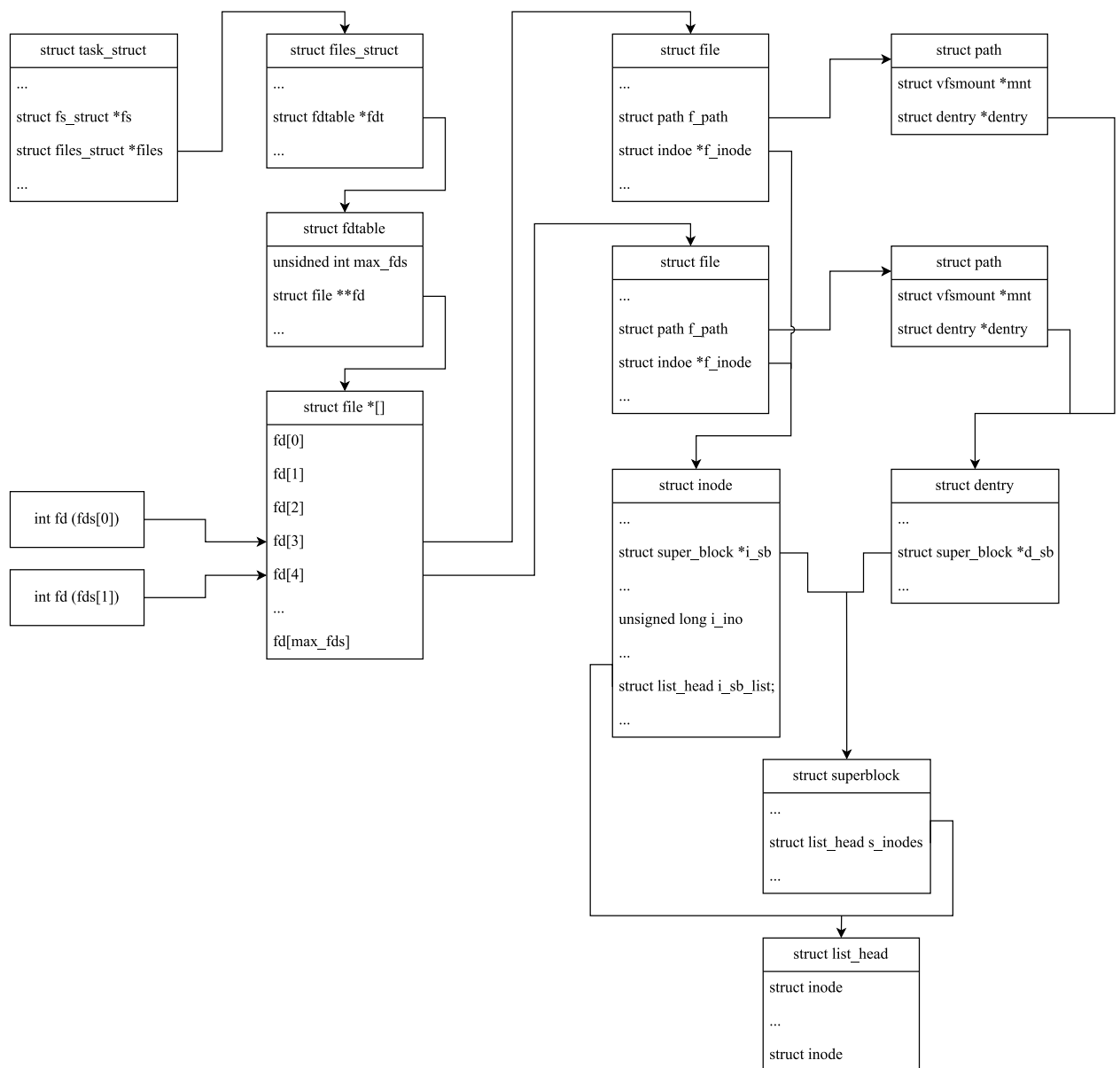
thread 1: a
thread 1: b
thread 1: c
thread 1: d
thread 1: e
thread 1: f
thread 1: g
thread 1: h
thread 1: i
thread 1: j
thread 1: k
thread 1: l
thread 1: m
thread 1: n
thread 1: o
thread 1: p
thread 1: q
thread 2: a
thread 1: r
thread 2: b
thread 1: s
thread 2: c
thread 1: t
thread 2: d
thread 1: u
thread 2: e

```

Рисунок 3.2 – Результат работы второй программы.

Аналогично однопоточной программе, потоки не мешают друг другу читать из файла, так как они получают независимые дескрипторы. Внутри одного потока все символы выводятся подряд в порядке следования, однако из-за асинхронности выполнения потоков, порядок вывода символов не определен.

3.3 Связь структур



4 Программа 3

4.1 Вариант 1 (небуферизованный ввод-вывод)

4.1.1 Однопоточная программа

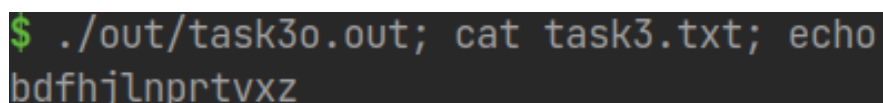
Листинг 9 – Третья программа 1 (однопоточный вариант)

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <fcntl.h>
4 #include <unistd.h>
5
6 #include "stat.h"
7
8 #define MAX_FDS 10
9
10 struct arg
11 {
12     int valid;
13     size_t amount;
14 };
15
16 int main(int argc, char **argv)
17 {
18     const struct arg arg = parse_args(argc, argv);
19
20     if (EXIT_SUCCESS != arg.valid)
21         return EXIT_FAILURE;
22
23     int fds[MAX_FDS] = {-1};
24     int rc = EXIT_SUCCESS;
25
26     for (size_t i = 0; EXIT_SUCCESS == rc && arg.amount > i; i++)
27     {
28         fds[i] = open("task3.txt", O_CREAT | O_TRUNC | O_WRONLY);
29         STAT_WRAP("open", print_path_stat("task3.txt"));
30
31         if (0 > fds[i])
32         {
33             perror("open error\n");
```

```

34         rc = EXIT_FAILURE;
35     }
36 }
37
38 if (EXIT_SUCCESS == rc)
39     for (char c = 'a'; 'z' >= c; c++)
40     {
41         write(fds[c % arg.amount], &c, sizeof(char));
42         STAT_WRAP("write", print_path_stat("task3.txt"));
43     }
44
45 for (size_t i = 0; arg.amount > i; i++)
46     if (0 <= fds[i])
47     {
48         close(fds[i]);
49         STAT_WRAP("close", print_path_stat("task3.txt"));
50     }
51
52 return EXIT_SUCCESS;
53 }

```



```

$ ./out/task3o.out; cat task3.txt; echo
bdfhjlnprtvxz

```

Рисунок 4.1 – Результат работы третьей программы 1.

Аналогично однопоточному варианту предыдущей программы, процесс получает несколько независимых дескрипторов одного файла. В результате этого, при поочередной записи символов, происходит затирание символов, расположенных на одной позиции. Полученный файл содержит запись, формируемую с использованием последнего файлового дескриптора.

4.1.2 Многопоточная программа

Листинг 10 – Третья программа 1 (многопоточный вариант)

```

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

```

```

6 | #include <string.h>
7 |
8 | #include "stat.h"
9 |
10 | #define MAX_THREADS 10
11 |
12 | struct arg
13 | {
14 |     int valid;
15 |     size_t threads;
16 | };
17 |
18 | struct data
19 | {
20 |     char c;
21 |     pthread_mutex_t mutex;
22 |     const struct arg *arg;
23 | };
24 |
25 | struct handle
26 | {
27 |     size_t id;
28 |     struct data *data;
29 | };
30 |
31 | void *thread_func(void *_arg)
32 | {
33 |     struct handle *arg = _arg;
34 |     char tmp[30];
35 |     ssize_t len;
36 |
37 |     int fd = open("task3.txt", O_WRONLY);
38 |     STAT_WRAP("open", print_path_stat("task3.txt"));
39 |
40 |     if (0 > fd)
41 |     {
42 |         perror("open error\n");
43 |
44 |         return NULL;
45 |     }
46 |
47 |     for (int run = 1, rc = EXIT_SUCCESS; EXIT_SUCCESS == rc && run;)
48 |     {
49 |         rc = pthread_mutex_lock(&arg->data->mutex);
50 |
51 |         if (EXIT_SUCCESS == rc)
52 |         {
53 |             if ('z' < arg->data->c)

```

```

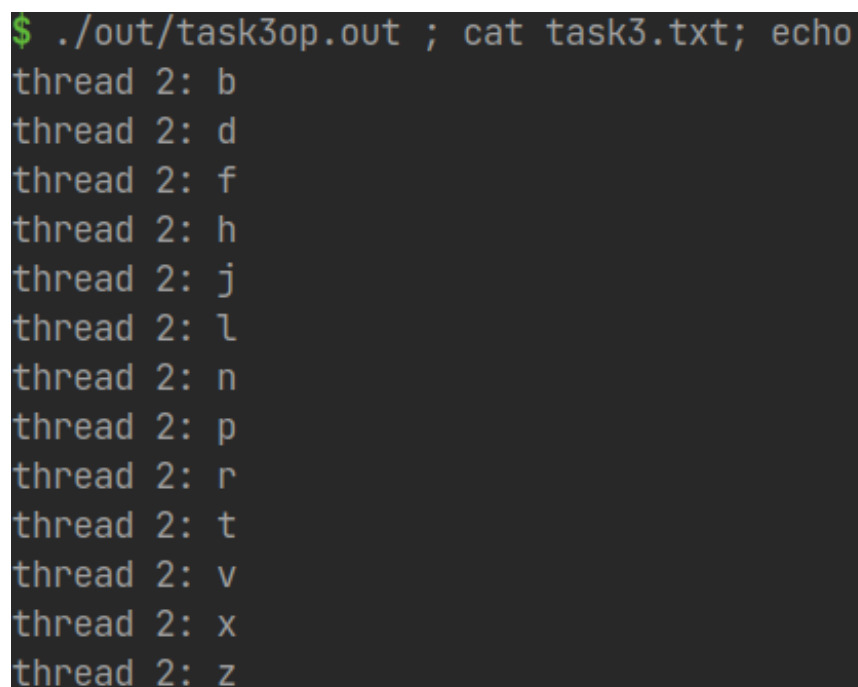
54         run = 0;
55     else if ((arg->data->c == 'a') % arg->data->arg->threads
56              == arg->id - 1)
57     {
58         len = sprintf(tmp, "thread %zu: %c\n", arg->id,
59                      arg->data->c++);
60         write(fd, tmp, len);
61         STAT_WRAP("write", print_path_stat("task3.txt"));
62     }
63
64     if (EXIT_SUCCESS != pthread_mutex_unlock(&arg->data->mutex))
65     {
66         perror("mutex_unlock error\n");
67         rc = EXIT_FAILURE;
68     }
69 }
70 else
71     perror("mutex_lock error\n");
72 }
73
74 close(fd);
75 STAT_WRAP("close", print_path_stat("task3.txt"));
76
77 return NULL;
78 }
79
80 int main(int argc, char **argv)
81 {
82     const struct arg arg = parse_args(argc, argv);
83
84     if (EXIT_SUCCESS != arg.valid)
85         return EXIT_FAILURE;
86
87     if (EXIT_SUCCESS != close(open("task3.txt", O_CREAT | O_TRUNC)))
88         return EXIT_FAILURE;
89
90     struct data data = {
91         .c = 'a',
92         .mutex = {{0}},
93         .arg = &arg
94     };
95
96     if (EXIT_SUCCESS != pthread_mutex_init(&data.mutex, NULL))
97     {
98         perror("mutex_init error\n");
99         return EXIT_FAILURE;
100     }
101

```

```

102     struct handle args[MAX_THREADS];
103     pthread_t threads[MAX_THREADS];
104     int rc = EXIT_SUCCESS;
105
106     for (size_t i = 0; EXIT_SUCCESS == rc && arg.threads > i; i++)
107     {
108         args[i].id = i + 1;
109         args[i].data = &data;
110
111         if (EXIT_SUCCESS
112             != pthread_create(threads + i, NULL, thread_func, args + i))
113         {
114             perror("pthread_create error\n");
115             rc = EXIT_FAILURE;
116         }
117     }
118
119     for (size_t i = 0; arg.threads > i; i++)
120         pthread_join(threads[i], NULL);
121
122     pthread_mutex_destroy(&data.mutex);
123
124     return EXIT_SUCCESS;
125 }

```



```

$ ./out/task3op.out ; cat task3.txt; echo
thread 2: b
thread 2: d
thread 2: f
thread 2: h
thread 2: j
thread 2: l
thread 2: n
thread 2: p
thread 2: r
thread 2: t
thread 2: v
thread 2: x
thread 2: z

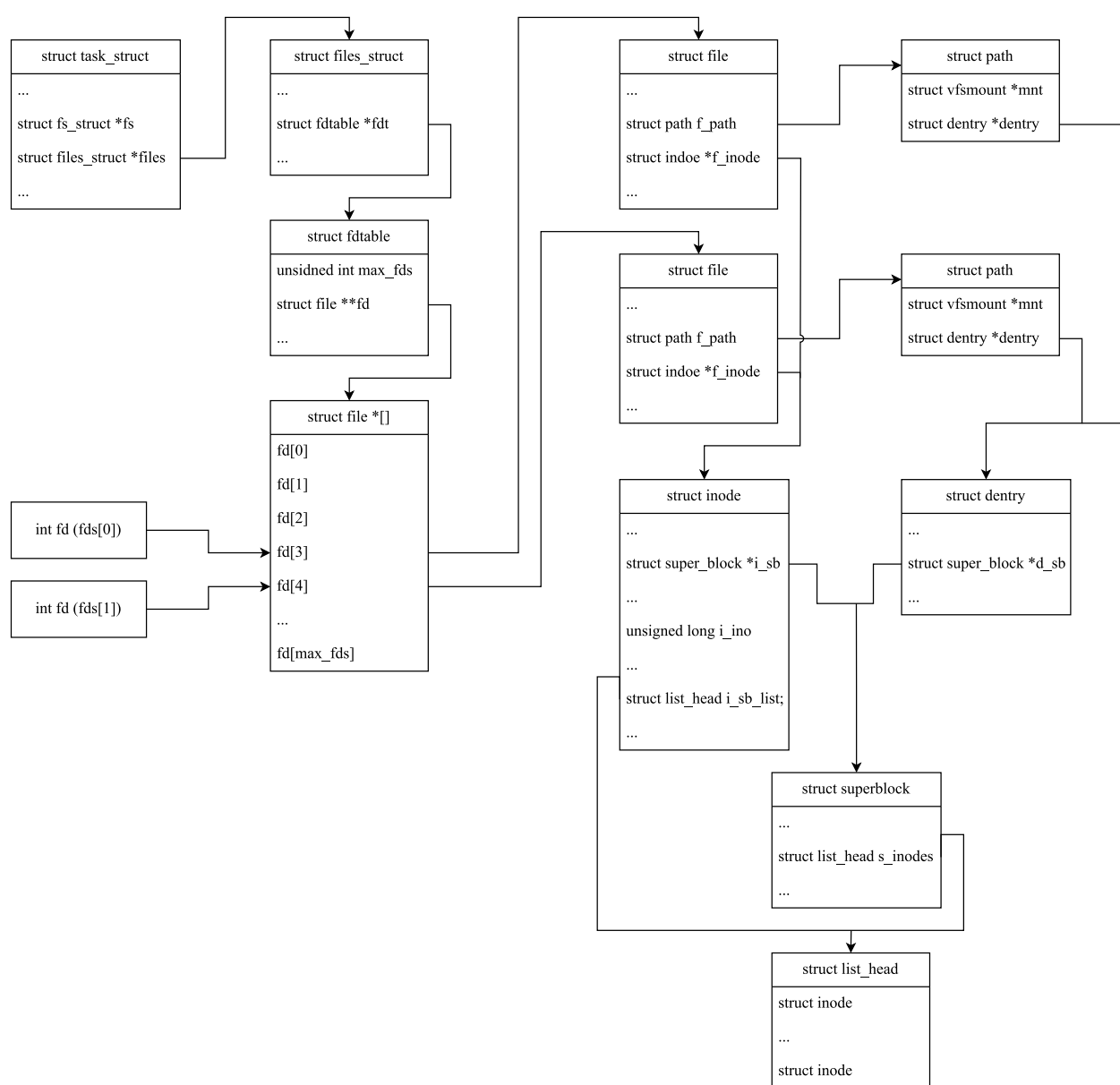
```

Рисунок 4.2 – Результат работы третьей программы 1.

Для обеспечения последовательной записи букв алфавита потоками в программе используется mutex, как средство синхронизации. Это приводит

к тому, что результат выполнения однопоточной и многопоточной версий программы не отличаются.

4.1.3 Связь структур



4.2 Вариант 2 (буферизованный ввод-вывод)

4.2.1 Однопоточная программа

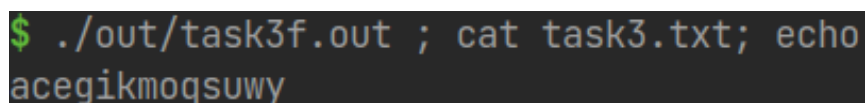
Листинг 11 – Третья программа 2 (однопоточный вариант)

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <fcntl.h>
4 #include <unistd.h>
5
6 #include "stat.h"
7
8 #define MAX_FDS 10
9
10 struct arg
11 {
12     int valid;
13     size_t amount;
14 };
15
16 int main(int argc, char **argv)
17 {
18     const struct arg arg = parse_args(argc, argv);
19
20     if (EXIT_SUCCESS != arg.valid)
21         return EXIT_FAILURE;
22
23     FILE *fds[MAX_FDS] = {NULL};
24     int rc = EXIT_SUCCESS;
25
26     for (size_t i = 0; EXIT_SUCCESS == rc && arg.amount > i; i++)
27     {
28         fds[i] = fopen("task3.txt", "w");
29         STAT_WRAP("fopen", print_path_stat("task3.txt"));
30
31         if (!fds[i])
32         {
33             perror("fopen error\n");
34             rc = EXIT_FAILURE;
35         }
36     }
37
38     if (EXIT_SUCCESS == rc)
```

```

39     for (char c = 'a'; 'z' >= c; c++)
40     {
41         fprintf(fds[c % arg.amount], "%c", c);
42         STAT_WRAP("fprintf", print_path_stat("task3.txt"));
43     }
44
45     for (size_t i = 0; arg.amount > i; i++)
46         if (fds[i])
47         {
48             fclose(fds[i]);
49             STAT_WRAP("fclose", print_path_stat("task3.txt"));
50         }
51
52     return rc;
53 }

```



```

$ ./out/task3f.out ; cat task3.txt; echo
acegikmoqsuwy

```

Рисунок 4.3 – Результат работы третьей программы 2.

Библиотека `stdio.h` предполагает, что запись буфера на диск возможна в одном из трех случаев:

- произошло заполнение буфера;
- произошло закрытие файла;
- произошел вызов функции `fflush` (принудительная запись).

Стандартный размер буфера на используемом компьютере составляет 8 КБ, из чего следует, что запись в программе будет происходить по закрытию файла.

Листинг 12 – Стандартный размер буфера в библиотеке `stdio`

```

1 /* Default buffer size. */
2 #define BUFSIZ 8192

```

Следовательно, содержимым файла будет результат работы с одним из библиотечных файловых дескрипторов.

4.2.2 Многопоточная программа

Листинг 13 – Вторая программа 2 (многопоточный вариант)

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <fcntl.h>
4 #include <unistd.h>
5 #include <pthread.h>
6 #include <string.h>
7
8 #include "stat.h"
9
10 #define MAX_THREADS 10
11
12 struct arg
13 {
14     int valid;
15     size_t threads;
16 };
17
18 struct data
19 {
20     char c;
21     pthread_mutex_t mutex;
22     const struct arg *arg;
23 };
24
25 struct handle
26 {
27     size_t id;
28     struct data *data;
29 };
30
31 void *thread_func(void *_arg)
32 {
33     struct handle *arg = _arg;
34
35     FILE *fd = fopen("task3.txt", "r+");
36     STAT_WRAP("fopen", print_path_stat("task3.txt"));
37
38     if (!fd)
39     {
40         perror("fopen error\n");
41
42         return NULL;
```

```

43     }
44
45     for (int run = 1, rc = EXIT_SUCCESS; EXIT_SUCCESS == rc && run;)
46     {
47         rc = pthread_mutex_lock(&arg->data->mutex);
48
49         if (EXIT_SUCCESS == rc)
50         {
51             if ('z' < arg->data->c)
52                 run = 0;
53             else if ((arg->data->c - 'a') % arg->data->arg->threads
54                     == arg->id - 1)
55             {
56                 fprintf(fd, "thread %zu: %c\n", arg->id, arg->data->c++);
57                 STAT_WRAP("fprintf", print_path_stat("task3.txt"));
58             }
59
60             if (EXIT_SUCCESS != pthread_mutex_unlock(&arg->data->mutex))
61             {
62                 perror("mutex_unlock error\n");
63                 rc = EXIT_FAILURE;
64             }
65         }
66         else
67             perror("mutex_lock error\n");
68     }
69
70     fclose(fd);
71     STAT_WRAP("fclose", print_path_stat("task3.txt"));
72
73     return NULL;
74 }
75
76
77 int main(int argc, char **argv)
78 {
79     const struct arg arg = parse_args(argc, argv);
80
81     if (EXIT_SUCCESS != arg.valid)
82         return EXIT_FAILURE;
83
84     if (EXIT_SUCCESS != close(open("task3.txt", O_CREAT | O_TRUNC)))
85         return EXIT_FAILURE;
86
87     struct data data = {
88         .c = 'a',
89         .mutex = {{0}},
90         .arg = &arg

```

```

91     };
92
93     if (EXIT_SUCCESS != pthread_mutex_init(&data.mutex, NULL))
94     {
95         perror("mutex_init error\n");
96         return EXIT_FAILURE;
97     }
98
99     struct handle args[MAX_THREADS];
100    pthread_t threads[MAX_THREADS];
101    int rc = EXIT_SUCCESS;
102
103    for (size_t i = 0; EXIT_SUCCESS == rc && arg.threads > i; i++)
104    {
105        args[i].id = i + 1;
106        args[i].data = &data;
107
108        if (EXIT_SUCCESS
109            != pthread_create(threads + i, NULL, thread_func, args + i))
110        {
111            perror("pthread_create error\n");
112            rc = EXIT_FAILURE;
113        }
114    }
115
116    for (size_t i = 0; arg.threads > i; i++)
117        pthread_join(threads[i], NULL);
118
119    pthread_mutex_destroy(&data.mutex);
120
121    return rc;
122 }

```

```

$ ./out/task3fp.out ; cat task3.txt; echo
thread 2: b
thread 2: d
thread 2: f
thread 2: h
thread 2: j
thread 2: l
thread 2: n
thread 2: p
thread 2: r
thread 2: t
thread 2: v
thread 2: x
thread 2: z

```

```

$ ./out/task3fp.out ; cat task3.txt; echo
thread 1: a
thread 1: c
thread 1: e
thread 1: g
thread 1: i
thread 1: k
thread 1: m
thread 1: o
thread 1: q
thread 1: s
thread 1: u
thread 1: w
thread 1: y

```

Рисунок 4.4 – Результат работы второй программы 2.

В отличие от однопоточной программы, порядок закрытия файлов в данной реализации не определен, следовательно, в зависимости от стечения обстоятельств файл может содержать результат работы одного из потоков.

4.2.3 Связь структур

