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, искусственный интеллект и системы управления**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии**»

Отчет по лабораторной работе №10

по курсу «Операционные системы»

«Буферизованный и не буферизованный ввод-вывод»

Студент группы **ИУ7-64Б** _____ **Д. С. Чепиго**

(Подпись, дата) (И.О. Фамилия)

Преподаватели _____ **Н. Ю. Рязанова**

(Подпись, дата) (И.О. Фамилия)

***2023** г.*

1 Используемые структуры

Версия ядра: 5.15.32.

Листинг 1 – struct _IO_FILE

```
struct _IO_FILE
{
int _flags; /* High-order word is _IO_MAGIC; rest is flags. */

/* The following pointers correspond
to the C++ streambuf protocol.*/

char *_IO_read_ptr;     /* Current read pointer */
char *_IO_read_end;     /* End of get area. */
char *_IO_read_base;     /* Start of putback+get area. */
char *_IO_write_base;     /* Start of put area. */
char *_IO_write_ptr;     /* Current put pointer. */
char *_IO_write_end;     /* End of put area. */
char *_IO_buf_base;     /* Start of reserve area. */
char *_IO_buf_end;     /* End of reserve area. */

/* The following fields are used
to support backing up and undo. */

/* Pointer to start of non-current get area */
char *_IO_save_base;
/*Pointer to first valid character of backup area */
char *_IO_backup_base;
/* Pointer to end of non-current get area. */
char *_IO_save_end;

struct _IO_marker *_markers;

struct _IO_FILE *_chain;
```

```
30
31  int _fileno;
32  int _flags2;
33
34  __off_t _old_offset;
35  /* This used to be _offset but it's too small. */
36
37  /* 1+column number of pbase(); 0 is unknown. */
38  unsigned short _cur_column;
39  signed char _vtable_offset;
40  char _shortbuf[1];
41
42  _IO_lock_t *_lock;
43  #ifdef _IO_USE_OLD_IO_FILE
44  };
```

Листинг 2 – struct file

```
1   struct file {
2   union {
3       struct llist_node   fu_llist;
4       struct rcu_head     fu_rcuhead;
5   } f_u;
6   struct path     f_path;
7   struct inode    *f_inode;    /* cached value */
8   const struct file_operations    *f_op;
9
10  /*
11  * Protects f_ep, f_flags.
12  * Must not be taken from IRQ context.
13  */
14  spinlock_t              f_lock;
15  enum rw_hint            f_write_hint;
16  atomic_long_t           f_count;
```

```
unsigned int            f_flags;
fmode_t                 f_mode;
struct mutex            f_pos_lock;
loff_t                  f_pos;
struct fown_struct      f_owner;
const struct cred      *f_cred;
struct file_ra_state    f_ra;

u64         f_version;
#ifdef CONFIG_SECURITY
void    *f_security;
#endif
/* needed for tty driver, and maybe others */
void        *private_data;

#ifdef CONFIG_EPOLL
/* Used by fs/eventpoll.c to link all the hooks to this file */
struct hlist_head    *f_ep;
#endif /* #ifdef CONFIG_EPOLL */
struct address_space    *f_mapping;
errseq_t    f_wb_err;
errseq_t    f_sb_err; /* for syncfs */
} __randomize_layout
/* lest something weird decides that 2 is OK */
__attribute__((aligned(4)));
```

Листинг 3 – struct stat

```
struct stat {
unsigned long    st_dev;    /* Device.  */
unsigned long    st_ino;    /* File serial number.  */
unsigned int    st_mode;    /* File mode.  */
unsigned int    st_nlink;    /* Link count.  */
unsigned int    st_uid;    /* User ID of the file's owner.  */
```

```
unsigned int    st_gid;    /* Group ID of the file's group. */
unsigned long    st_rdev;    /* Device number, if device.  */
unsigned long    __pad1;
long            st_size;    /* Size of file, in bytes.  */
int             st_blksize;    /* Optimal block size for I/O. */
int            __pad2;
/* Number 512-byte blocks allocated.*/
long            st_blocks;
long            st_atime;    /* Time of last access.  */
unsigned long    st_atime_nsec;
long            st_mtime;    /* Time of last modification.  */
unsigned long    st_mtime_nsec;
long            st_ctime;    /* Time of last status change.  */
unsigned long    st_ctime_nsec;
unsigned int    __unused4;
unsigned int    __unused5;
};
```

2 Первая программа

2.1 Базовый вариант

Листинг 4 – Первая программа, базовый вариант

```c
#include <stdio.h>
#include <fcntl.h>

int main(void)
{
    int fd = open("alphabet.txt", O_RDONLY);
    FILE *fs1 = fdopen(fd, "r");
    char buff1[20];
    setvbuf(fs1, buff1, _IOFBF, 20);
    FILE *fs2 = fdopen(fd, "r");
    char buff2[20];
    setvbuf(fs2, buff2, _IOFBF, 20);

    int flag1 = 1, flag2 = 2;
    while (flag1 == 1 || flag2 == 1)
    {
        char c;
        flag1 = fscanf(fs1, "%c", &c);
        if (flag1 == 1)
            fprintf(stdout, "%c", c);
        flag2 = fscanf(fs2, "%c", &c);
        if (flag2 == 1)
            fprintf(stdout, "%c", c);
    }

    return 0;
}
```

Рисунок 1 – Вывод программы

С помощью системного вызова open() создается дескриптор открытого файла (только для чтения). Системный вызов open() возвращает индекс в массиве fd структуры files_struct. Библиотечная функция fdopen() возвращает указатели на struct FILE (fs1 и fs2), которые ссылаются на дескриптор, созданный системным вызовом open(). Далее создаются буферы buff1 и buff2 размером 20 байт. Для дескрипторов fs1 и fs2 функцией setvbuf() задаются соответствующие буферы и тип буферизации _IOFBF.

Далее fscanf() выполняется в цикле поочерёдно для fs1 и fs2. Так как установлена полная буферизация, то при первом вызове fscanf() буфер будет заполнен полностью либо вплоть до конца файла, а f_pos установится на следующий за последним записанным в буфер символ.

При первом вызове fscanf() для fs1 в буфер buff1 считаются первые 20 символов (abcdefghijklmnopqrst). Значение f_pos в структуре struct_file открытого увеличится на 20. В переменную с записывается символ ’a’ и выводится с помощью fprintf(). При первом вызове fscanf() для fs2 в буфер buff2 считываются оставшиеся в файле символы — uvwxyz (в переменную с записывается символ ’u’).

В цикле символы из buff1 и buff2 будут поочередно выводиться до тех пор, пока символы в одном из буферов не закончатся. Тогда на экран будут последовательно выведены оставшиеся символы из другого буфера.

2.2 Связи структур

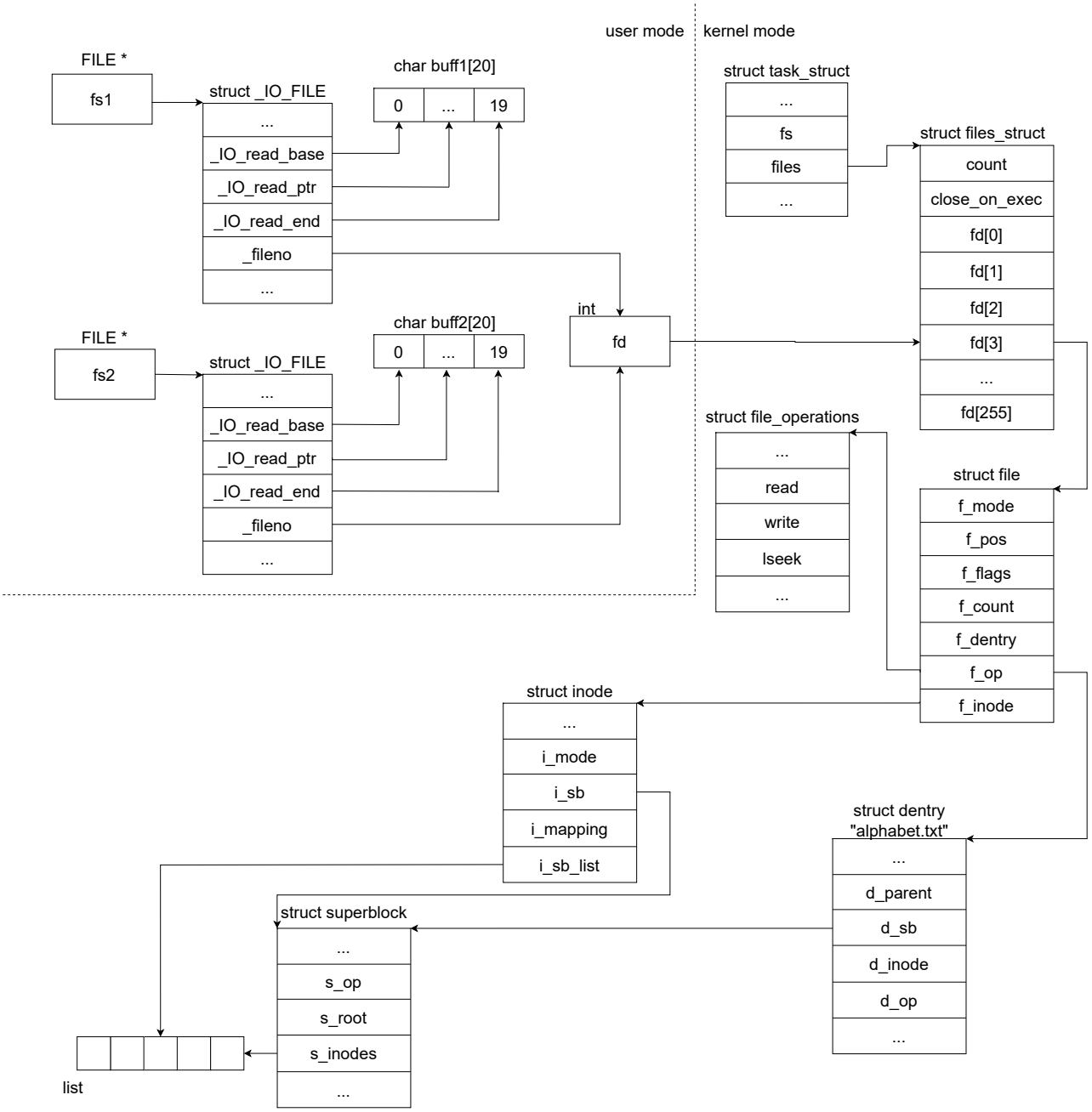

Рисунок 2 – Связи структур в первой программе

2.3 Многопоточный вариант

Листинг 5 – Первая программа с созданием двух дополнительных потоков

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <pthread.h>

void *thread_func1(void *fs)
{
    char c;
    while (fscanf((FILE *)fs, "%c", &c) == 1)
        fprintf(stdout, "subthread 1:   %c\n", c);
    return NULL;
}

void *thread_func2(void *fs)
{
    char c;
    while (fscanf((FILE *)fs, "%c", &c) == 1)
        fprintf(stdout, "subthread 2:   %c\n", c);
    return NULL;
}

int main(void)
{
    int fd = open("alphabet.txt", O_RDONLY);
    FILE *fs[2] = {fdopen(fd, "r"), fdopen(fd, "r")};
    char buff[2][20];
    setvbuf(fs[0], buff[0], _IOFBF, 20);
    setvbuf(fs[1], buff[1], _IOFBF, 20);
    char c;
    pthread_t threads[2];
    void *(*thread_funcs[2])(void *) =
```

```
32          {thread_func1, thread_func2};
33
34      for (size_t i = 0; i < 2; i++)
35          if (pthread_create(&threads[i], NULL,
36                             thread_funcs[i], fs[i]) != 0)
37          {
38              perror("pthread_create\n");
39              exit(1);
40          }
41
42      for (size_t i = 0; i < 2; i++)
43          if (pthread_join(threads[i], NULL) != 0)
44          {
45              perror("pthread_join\n");
46              exit(1);
47          }
48
49      return 0;
50  }
```

```
[d.chepigo@d-chepigo ~/D/b/b/b/6/l/r/code (main)> ./a.out
subthread 1:   a
subthread 1:   b
subthread 2:   u
subthread 2:   v
subthread 2:   w
subthread 2:   x
subthread 2:   y
subthread 2:   z
subthread 1:   c
subthread 1:   d
subthread 1:   e
subthread 1:   f
subthread 1:   g
subthread 1:   h
subthread 1:   i
subthread 1:   j
subthread 1:   k
subthread 1:   l
subthread 1:   m
subthread 1:   n
subthread 1:   o
subthread 1:   p
subthread 1:   q
subthread 1:   r
subthread 1:   s
subthread 1:   t
```

Рисунок 3 – Вывод программы

В однопоточной программе в цикле поочередно выводятся символы из buff1 и buff2, в то время как в многопоточной программе главный поток начинает вывод раньше, так как для дополнительного потока сначала затрачи- вается время на его создание, и только потом начинается вывод. При создании дополнительных потоков связи структур не изменяются, так как ресурсами (в том числе и открытыми файлами) владеет процесс.

3 Вторая программа

3.1 Базовый вариант

Листинг 6 – Вторая программа, базовый вариант

```c
#include <fcntl.h>
#include <unistd.h>

int main()
{
    char c;
    int fd1 = open("alphabet.txt", O_RDONLY);
    int fd2 = open("alphabet.txt", O_RDONLY);

    while (read(fd1, &c, 1) == 1 && read(fd2, &c, 1) == 1)
    {
        write(1, &c, 1);
        write(1, &c, 1);
    }

    return 0;
}
```

```
[d.chepigo@d-chepigo ~/D/b/b/b/6/l/r/code (main)> ./a.out
aabbccddeeffgghhiijjkkllmmnnooppqqrrssttuuvvwwxxyyzz
```

Рисунок 4 – Вывод программы

В программе один и тот же файл открывается 2 раза для чтения. При выполнении системного вызова open() создаётся дескриптор открытого файла в таблице открытых файлов процесса и запись в системной таблице открытых файлов. Так как файл открывается 2 раза, то в системной таблице открытых файлов будет создано 2 дескриптора struct file, каждый из которых имеет соб-

ственный указатель f_pos. По этой причине чтение становится независимым — при вызове read() для обоих дескрипторов по очереди, оба указателя проходят по всем позициям файла, и каждый символ считывается и выводится по два раза. При этом оба дескриптора struct file ссылаются на один и тот же inode.

3.2 Связи структур

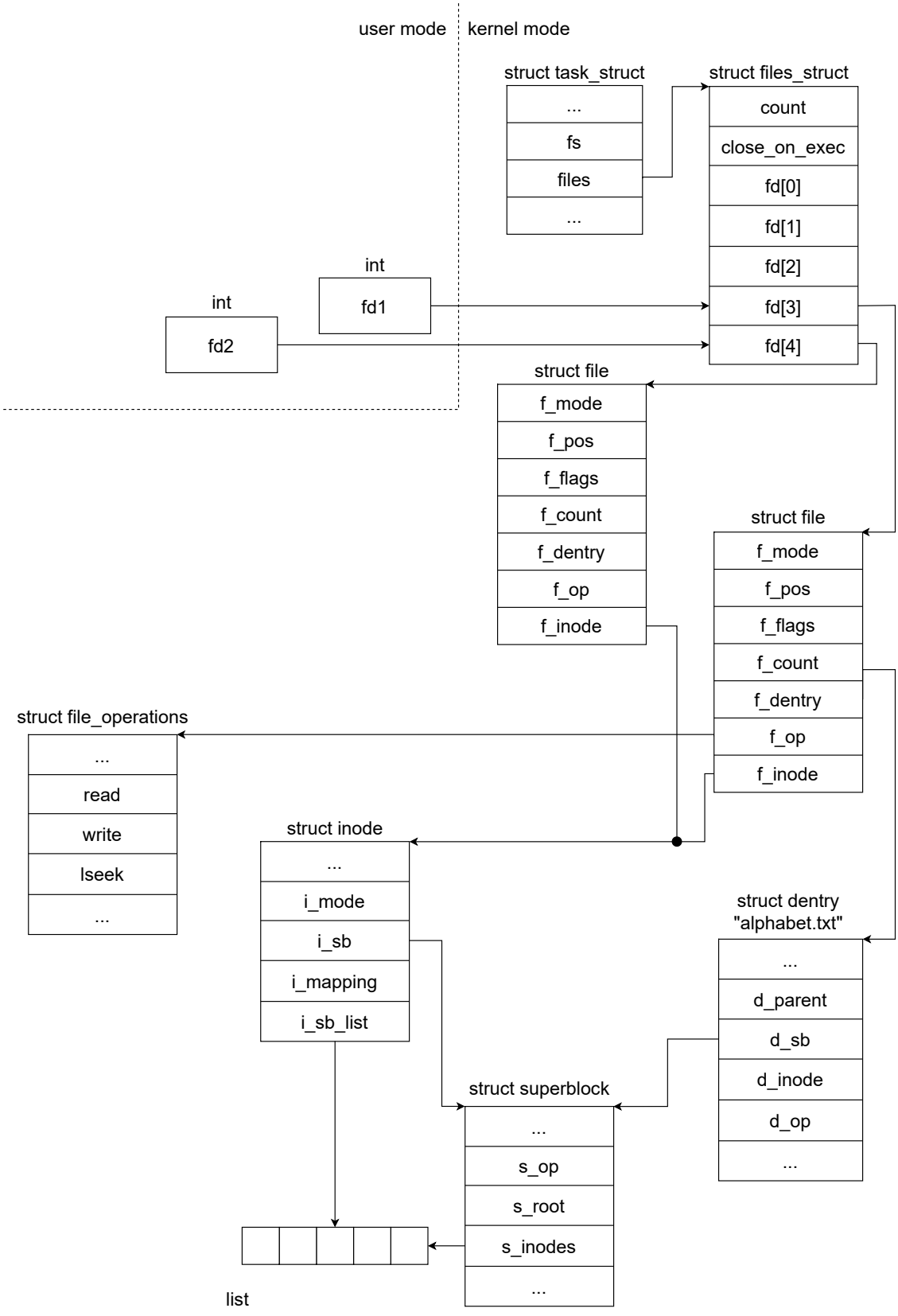

Рисунок 5 – Связи структур во второй программе

3.3 Многопоточный вариант

Листинг 7 – Вторая программа с созданием двух дополнительных потоков

```
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <pthread.h>
#include <stdio.h>

void *thread_func1(void *fd)
{
    char c;
    while (read((int)fd, &c, 1) == 1)
        write(1, &c, 1);

    return NULL;
}

void *thread_func2(void *fd)
{
    char c;
    while (read((int)fd, &c, 1) == 1)
        write(1, &c, 1);

    return NULL;
}

int main(void)
{
    int fd[2] = {open("alphabet.txt", O_RDONLY),
                 open("alphabet.txt", O_RDONLY)};
    char c;
    pthread_t threads[2];
    void *(*thread_funcs[2])(void *) =
```

```
                {thread_func1, thread_func2};

    for (size_t i = 0; i < 2; i++)
        if (pthread_create(&threads[i], NULL,
                    thread_funcs[i], fd[i]) != 0)
        {
            perror("pthread_create\n");
            exit(1);
        }

    for (size_t i = 0; i < 2; i++)
        if (pthread_join(threads[i], NULL) != 0)
        {
            perror("pthread_join\n");
            exit(1);
        }

    return 0;
}
```

```
[d.chepigo@d-chepigo ~/D/b/b/b/6/l/r/code (main)> ./a.out
abcdefagbhcidjklmnopqrstuevwfxgyhzijklmnopqrstuvwxyz
```

Рисунок 6 – Вывод программы

В однопоточной программе в цикле каждый символ из файла выводится два раза подряд, а в многопоточной программе порядок вывода символов не определён, так как потоки выполняются параллельно. При этом дополнительный поток начинает вывод позже главного, так как затрачивается время на его создание.

При создании дополнительных потоков связи структур не изменяются, так как ресурсами (в том числе и открытыми файлами) владеет процесс.

4 Вторая программа, второй вариант

4.1 Базовый вариант

Листинг 8 – Вторая программа, второй вариант, базовый вариант

```c
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>

struct stat statbuf;

#define PRINT_STAT(action) \
    do { \
        stat("q.txt", &statbuf); \
        fprintf(stdout, action ": inode number = %ld, \
                    size = %ld bytes, blksize = %ld\n", \
                   statbuf.st_ino, statbuf.st_size, \
                   statbuf.st_blksize); \
    } while (0)

int main()
{
    FILE *fs1 = fopen("q.txt", "w");
    PRINT_STAT("fopen fs1");
    FILE *fs2 = fopen("q.txt", "w");
    PRINT_STAT("fopen fs2");
    for (char c = 'a'; c <= 'z'; c++)
    {
        c % 2 ? fprintf(fs1, "%c", c) : fprintf(fs2, "%c", c);
        PRINT_STAT("fprintf");
    }
    fclose(fs1);
    PRINT_STAT("fclose fs1");
    fclose(fs2);
```

```c
    PRINT_STAT("fclose fs2");
    return 0;
}
```

```
d.chepigo@d-chepigo ~/D/b/b/b/6/l/r/code (main)> ./a.out
fopen fs1: inode number = 24350212, size = 0 bytes, blksize = 4096
fopen fs2: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fprintf: inode number = 24350212, size = 0 bytes, blksize = 4096
fclose fs1: inode number = 24350212, size = 13 bytes, blksize = 4096
fclose fs2: inode number = 24350212, size = 13 bytes, blksize = 4096
d.chepigo@d-chepigo ~/D/b/b/b/6/l/r/code (main)> cat q.txt
bdfhjlnprtvxz
```

Рисунок 7 – Вывод программы

В программе файл дважды открывается на запись функцией fopen() из библиотеки stdio.h. В системной таблице открытых файлов создаётся два дескриптора struct file, каждый из которых имеет собственный указатель f_pos, но оба ссылаются на один и тот же inode. С помощью библиотечной функции fprintf() выполняется буферизованный вывод. Буфер создается без явного указания. Существует 3 причины, по которым данные из буфера записываются в

файл:

1. Буфер заполнен.
2. Вызвана функция fflush() — принудительная запись.
3. Вызвана функция close()/fclose().

В данном случае запись в файл происходит в результате вызова функции fclose(). При вызове fclose() для fs1 буфер для fs1 записывается в файл. При вызове fclose() для fs2, все содержимое файла очищается, а в файл записывается содержимое буфера для fs2. В итоге произошла потеря данных, в файле окажется только содержимое буфера для fs2.

4.2 Связи структур

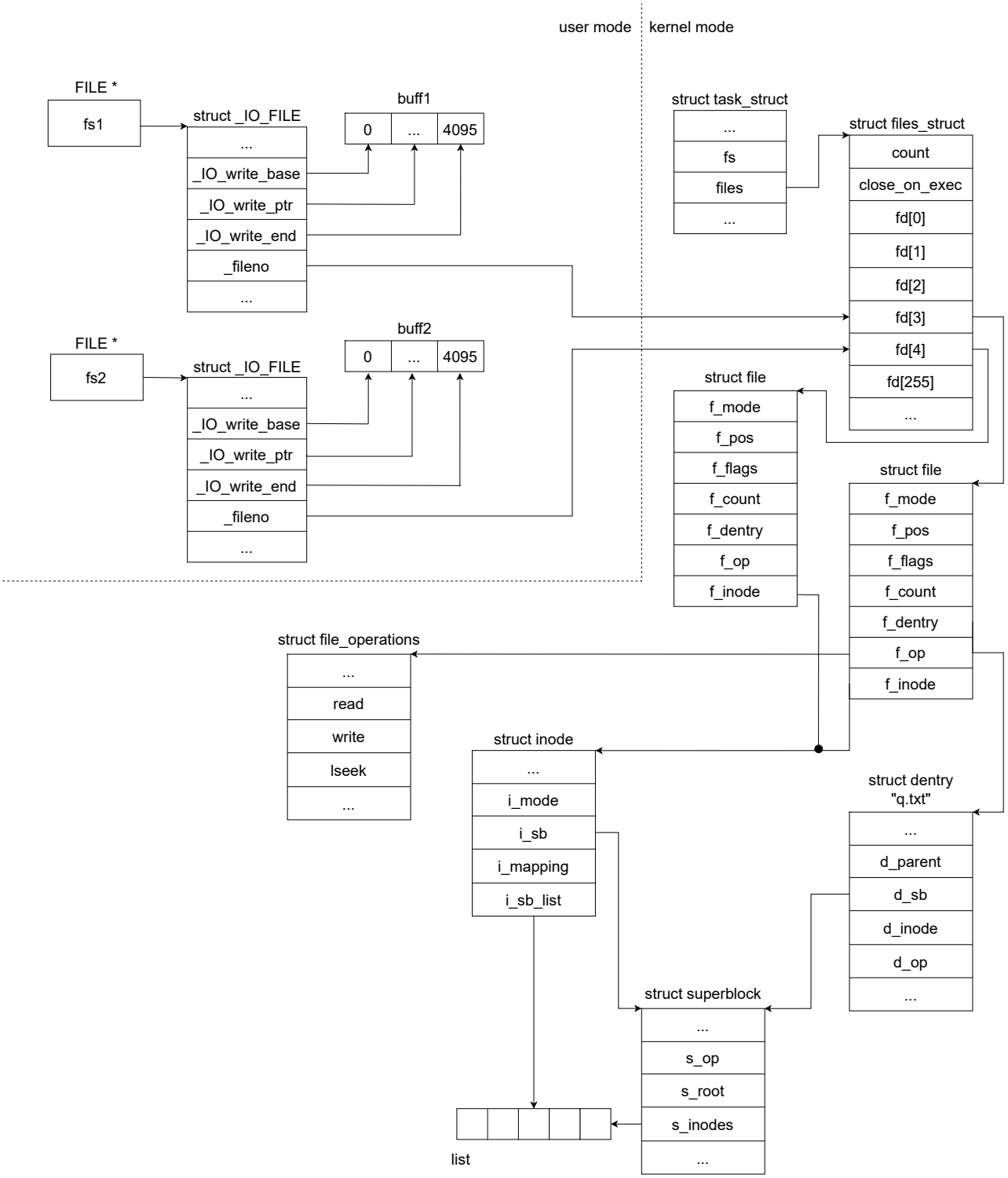

Рисунок 8 – Связи структур во втором варианте второй программе

4.3 Многопоточный вариант

Листинг 9 – Второй вариант второй программы с созданием двух дополнительных потоков

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <pthread.h>

void *thread_func1(void *fs)
{
    for (char c = 'a'; c <= 'z'; c += 2)
        fprintf((FILE *)fs, "subthread 1: %c\n", c);

    return NULL;
}

void *thread_func2(void *fs)
{
    for (char c = 'b'; c <= 'z'; c += 2)
        fprintf((FILE *)fs, "subthread 2: %c\n", c);

    return NULL;
}

int main(void)
{
    FILE *fs[2] = {fopen("q.txt", "w"), fopen("q.txt", "w")};
    pthread_t threads[2];
    void *(*thread_funcs[2])(void *)
                    = {thread_func1, thread_func2};

    for (size_t i = 0; i < 2; i++)
        if (pthread_create(&threads[i], NULL,
```

```
                        thread_funcs[i], fs[i]) != 0)
        {
            perror("pthread_create\n");
            exit(1);
        }

    for (size_t i = 0; i < 2; i++)
        if (pthread_join(threads[i], NULL) != 0)
        {
            perror("pthread_join\n");
            exit(1);
        }

    fclose(fs[0]);
    fclose(fs[1]);

    return 0;
}
```

```
[d.chepigo@d-chepigo ~/D/b/b/b/6/l/r/code (main)> ./a.out
[d.chepigo@d-chepigo ~/D/b/b/b/6/l/r/code (main)> cat q2.txt
 subthread 2: b
 subthread 2: d
 subthread 2: f
 subthread 2: h
 subthread 2: j
 subthread 2: l
 subthread 2: n
 subthread 2: p
 subthread 2: r
 subthread 2: t
 subthread 2: v
 subthread 2: x
 subthread 2: z
```

Рисунок 9 – Вывод программы

В многопоточной программе работа с файлом производится аналогично однопоточной программе. Если вызывать fclose() в дополнительном потоке, то порядок вывода символов будет не определён, так как нельзя предсказать заранее, какой поток последним вызовет fclose().

При создании дополнительных потоков связи структур не изменяются, так как ресурсами (в том числе и открытыми файлами) владеет процесс.

5 Третья программа

5.1 Базовый вариант

Листинг 10 – Третья программа, базовый вариант

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>

struct stat statbuf;

#define PRINT_STAT(action) \
    do { \
        stat("q.txt", &statbuf); \
        fprintf(stdout, action ": inode number = %ld, \
                    size = %ld bytes, blksize = %ld\n", \
                   statbuf.st_ino, statbuf.st_size, \
                   statbuf.st_blksize); \
    } while (0)

int main()
{
    int fd1 = open("q.txt", O_CREAT | O_WRONLY);
    PRINT_STAT("open fd1");
    int fd2 = open("q.txt", O_CREAT | O_WRONLY);
    PRINT_STAT("open fd2");
    for (char c = 'a'; c <= 'z'; c++)
    {
        c % 2 ? write(fd1, &c, 1) : write(fd2, &c, 1);
        PRINT_STAT("write");
    }
    close(fd1);
    PRINT_STAT("close fd1");
```

```
30      close(fd2);
31      PRINT_STAT("close fd2");
32      return 0;
33  }
```

```
[d.chepigo@d-chepigo ~/D/b/b/b/6/l/r/code (main)> ./a.out
open fd1: inode number = 24350212, size = 0 bytes, blksize = 4096
open fd2: inode number = 24350212, size = 0 bytes, blksize = 4096
write: inode number = 24350212, size = 1 bytes, blksize = 4096
write: inode number = 24350212, size = 1 bytes, blksize = 4096
write: inode number = 24350212, size = 2 bytes, blksize = 4096
write: inode number = 24350212, size = 2 bytes, blksize = 4096
write: inode number = 24350212, size = 3 bytes, blksize = 4096
write: inode number = 24350212, size = 3 bytes, blksize = 4096
write: inode number = 24350212, size = 4 bytes, blksize = 4096
write: inode number = 24350212, size = 4 bytes, blksize = 4096
write: inode number = 24350212, size = 5 bytes, blksize = 4096
write: inode number = 24350212, size = 5 bytes, blksize = 4096
write: inode number = 24350212, size = 6 bytes, blksize = 4096
write: inode number = 24350212, size = 6 bytes, blksize = 4096
write: inode number = 24350212, size = 7 bytes, blksize = 4096
write: inode number = 24350212, size = 7 bytes, blksize = 4096
write: inode number = 24350212, size = 8 bytes, blksize = 4096
write: inode number = 24350212, size = 8 bytes, blksize = 4096
write: inode number = 24350212, size = 9 bytes, blksize = 4096
write: inode number = 24350212, size = 9 bytes, blksize = 4096
write: inode number = 24350212, size = 10 bytes, blksize = 4096
write: inode number = 24350212, size = 10 bytes, blksize = 4096
write: inode number = 24350212, size = 11 bytes, blksize = 4096
write: inode number = 24350212, size = 11 bytes, blksize = 4096
write: inode number = 24350212, size = 12 bytes, blksize = 4096
write: inode number = 24350212, size = 12 bytes, blksize = 4096
write: inode number = 24350212, size = 13 bytes, blksize = 4096
write: inode number = 24350212, size = 13 bytes, blksize = 4096
close fd1: inode number = 24350212, size = 13 bytes, blksize = 4096
close fd2: inode number = 24350212, size = 13 bytes, blksize = 4096
[d.chepigo@d-chepigo ~/D/b/b/b/6/l/r/code (main)> cat q.txt
bdfhjlnprtvxz
```

Рисунок 10 – Вывод программы

В программе файл дважды открывается на запись функцией open(). В системной таблице открытых файлов создаётся два дескриптора struct file, каждый из которых имеет собственный указатель f_pos, но оба ссылаются на один и тот же inode. С помощью системного вызова write() выполняется небуферизо-

ванный вывод. При изменении порядка вызова функций close() вывод программы не изменяется, так как вывод не буферизуется.

Чтобы вывести алфавит полностью, можно для второго открытия файла использовать open() с флагом O_APPEND. В таком случае перед каждым вызовом write() для fd2 указатель f_pos будет устанавливаться в конце файла, как если бы использовался lseek().

5.2 Связи структур

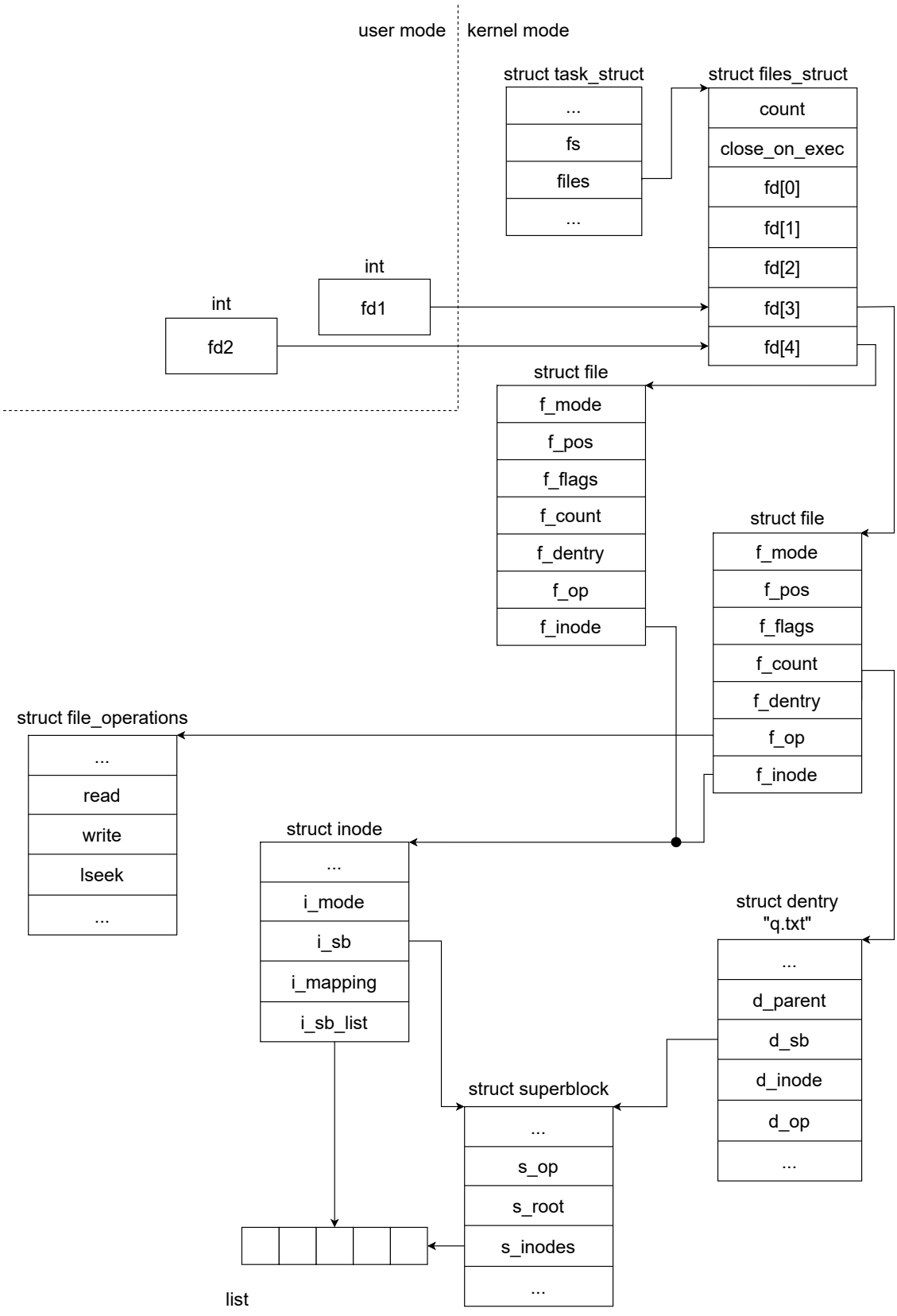

Рисунок 11 – Связи структур в первой программе

5.3 Многопоточный вариант

Листинг 11 – Первая программа с созданием двух дополнительных потоков

```
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <pthread.h>
#include <stdio.h>
#include <sys/stat.h>


struct stat statbuf;

#define PRINT_STAT(action) \
    do { \
        stat("q.txt", &statbuf); \
        fprintf(stdout, action ": inode number = %ld, \
                     size = %ld bytes, blksize = %ld\n", \
                    statbuf.st_ino, statbuf.st_size, \
                    statbuf.st_blksize); \
    } while (0)

pthread_mutex_t mx;

void *thread_func1(void *fd)
{
    for (char c = 'a'; c <= 'm'; c++)
    {
        pthread_mutex_lock(&mx);
        write((int)fd, &c, 1);
        PRINT_STAT("write");
        pthread_mutex_unlock(&mx);
    }

```

```
32      return NULL;
33  }
34
35  void *thread_func2(void *fd)
36  {
37      for (char c = 'n'; c <= 'z'; c++)
38      {
39          pthread_mutex_lock(&mx);
40          write((int)fd, &c, 1);
41          PRINT_STAT("write");
42          pthread_mutex_unlock(&mx);
43      }
44
45      return NULL;
46  }
47
48  int main(void)
49  {
50      int fd[2] = {open("q.txt", O_CREAT | O_WRONLY),
51                   open("q.txt", O_CREAT | O_WRONLY | O_APPEND)};
52      pthread_t threads[2];
53      void *(*thread_funcs[2])(void *)
54                      = {thread_func1, thread_func2};
55
56      if (pthread_mutex_init(&mx, NULL) != 0)
57      {
58          perror("pthread_mutex_init\n");
59          exit(1);
60      }
61
62      for (size_t i = 0; i < 2; i++)
63          if (pthread_create(&threads[i], NULL,
64                      thread_funcs[i], fd[i]) != 0)
65          {
66              perror("pthread_create\n");
```

```
67              exit(1);
68          }
69
70      for (size_t i = 0; i < 2; i++)
71          if (pthread_join(threads[i], NULL) != 0)
72          {
73              perror("pthread_join\n");
74              exit(1);
75          }
76
77      if (pthread_mutex_destroy(&mx) != 0)
78      {
79          perror("pthread_mutex_destroy\n");
80          exit(1);
81      }
82
83      close(fd[0]);
84      close(fd[1]);
85
86      return 0;
87  }
```

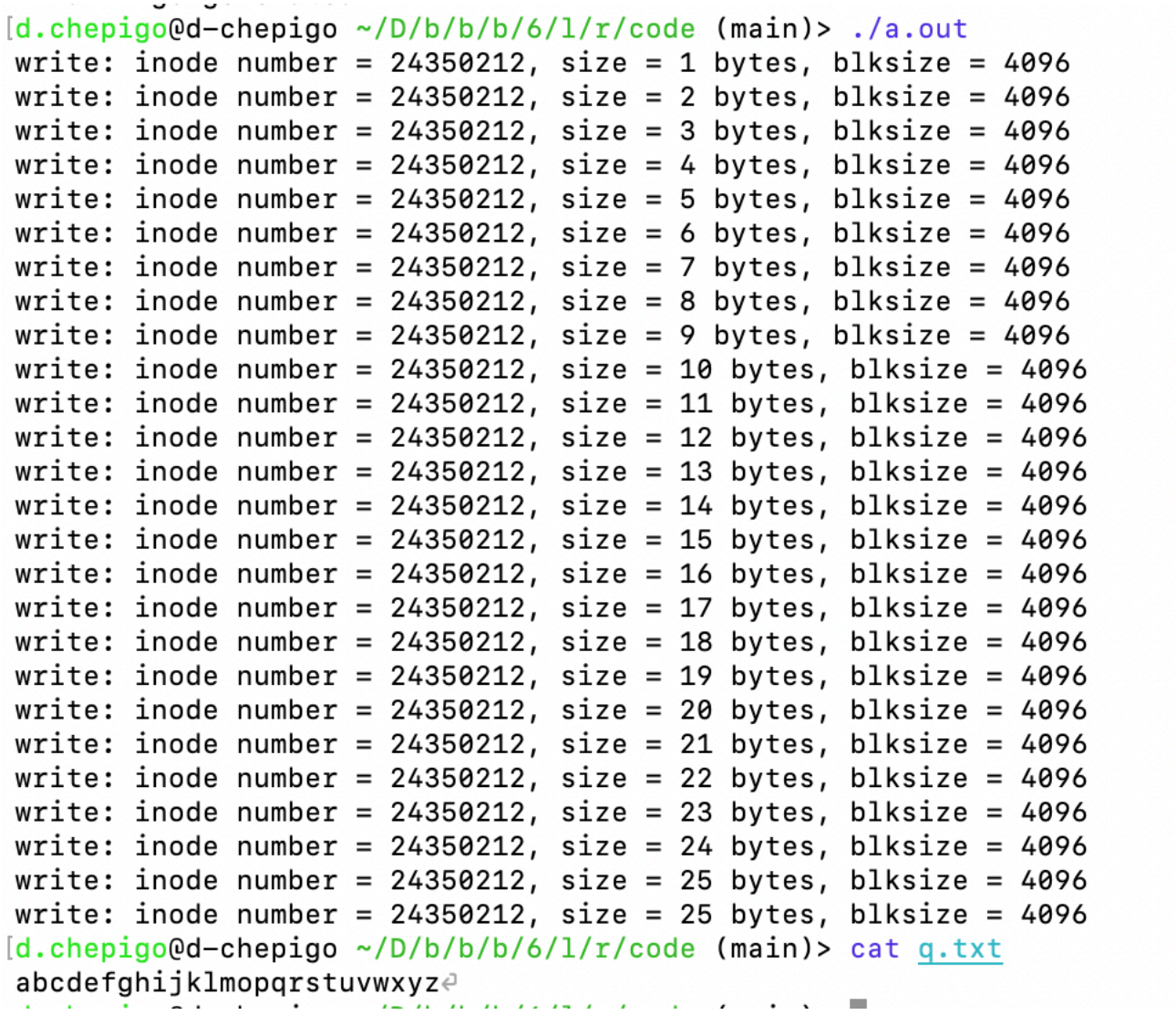

Рисунок 12 – Вывод программы

В многопоточной программе работа с файлом производится аналогично однопоточной программе. Если не использовать средства взаимоисключения (например, мьютекс), то вторая половина алфавита будет записываться частично, и поведение программы будет не определено.

При создании дополнительных потоков связи структур не изменяются, так как ресурсами (в том числе и открытыми файлами) владеет процесс.