



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
по курсу «Операционные системы»
на тему: «Буферизованный и не буферизованный ввод-вывод»

Студент ИУ7-63Б
(Группа)

(Подпись, дата)

Лысцев Н. Д.
(И. О. Фамилия)

Преподаватель

(Подпись, дата)

Рязанова Н. Ю.
(И. О. Фамилия)

2024 г.

1 Структуры

Версия ядра: 6.5.0-32-generic.

Листинг 1.1 – Структура struct `_IO_FILE` (начало)

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

    struct _IO_marker *_markers;

    struct _IO_FILE *_chain;

    int _fileno;
    int _flags2;
    __off_t _old_offset; /* This used to be _offset but it's too
                          small. */
}
```

Листинг 1.2 – Структура struct _IO_FILE (конец)

```
/* 1+column number of pbase(); 0 is unknown. */
unsigned short _cur_column;
signed char _vtable_offset;
char _shortbuf[1];

_IO_lock_t *_lock;
#ifdef _IO_USE_OLD_IO_FILE
};
```

Листинг 1.3 – Структура 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;
    long          st_blocks;  /* Number 512-byte blocks allocated. */
    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;
};
```

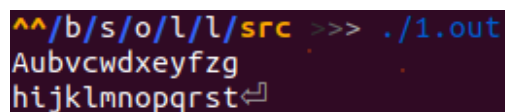
2 Программы

2.1 Первая программа

Листинг 2.1 – Первая программа

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

int main()
{
    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 = 1;
    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;
}
```



```
^^/b/s/o/l/l/src >>> ./1.out
Aubvcwdxeyfzg
hijklmnopqrst
```

Рисунок 2.1 — Результат выполнения программы

В первой программе файл открывается только для чтения (`O_RDONLY`). Системный вызов `open()` создает дескриптор открытого файла. Системный вызов `open()` возвращает индекс элемента в массиве `fd_array` структуры `files_struct`. Индекс равен 3, так как элементы массива `fd_array` с индексами 0, 1, 2 инициализированы стандартными потоками `stdin`, `stdout`, `stderr`. Библиотечная функция `fdopen()` возвращает указатели на структуры `struct FILE` (`fs1`, `fs2`), поля которых указывают на дескриптор `fd`, созданный системным вызовом `open()`. Создаются буферы `buff1`, `buff2` размером 20 байт. Функция `setvbuf()` для дескрипторов `fs1`, `fs2` задает буферы `buff1`, `buff2` с типом буферизации `_IOFBF`.

При первом вызове `fscanf()` для `fs1` в буфер `buff1` считываются первые 20 символов. Значение `f_pos` в структуре `struct _file` открытого файла увеличивается на 20. В переменную `s` записывается символ `'a'`, значение переменной `s` выводится на экран функцией `fprintf()`. При первом вызове `fscanf()` для `fs2` в буфер `buff2` считываются оставшиеся в файле символы. В переменную `s` записывается символ `'u'`, значение переменной `s` выводится на экран функцией `fprintf()`.

В цикле символы из `buff1`, `buff2` будут поочередно выводиться на экран, пока один из буферов не будет пуст. В этом случае оставшиеся символы из второго буфера будут последовательно выведены на экран.

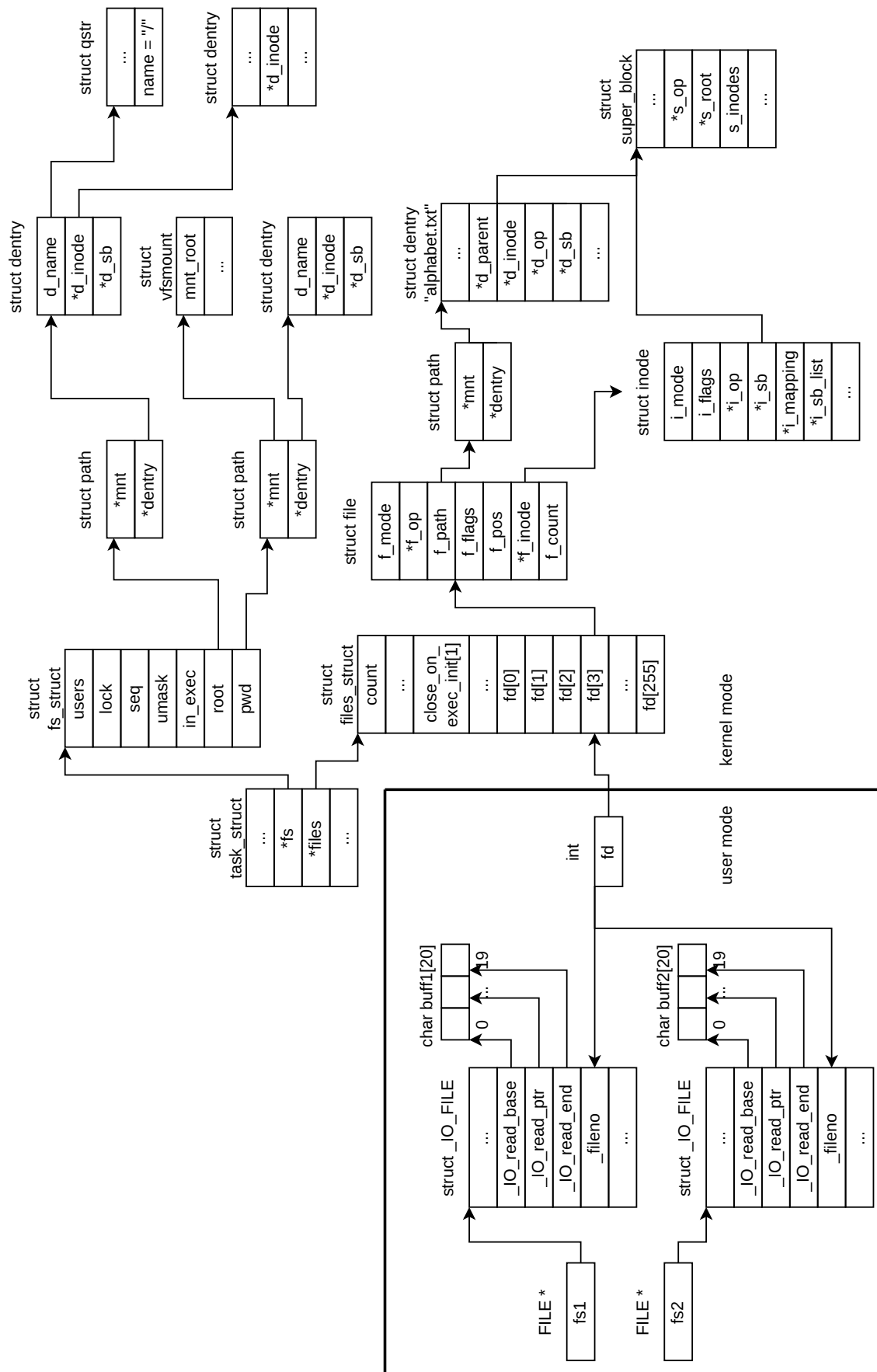


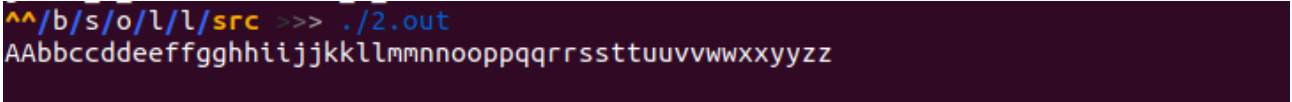
Рисунок 2.2 — Связи структур в первой программе

2.2 Вторая программа, первый вариант

2.2.1 Без использования потоков

Листинг 2.2 – Вторая программа, первый вариант

```
#include <fcntl.h>
#include <unistd.h>
int main()
{
    char c;
    int fd1 = open("alphabet.txt", O_RDONLY);
    int fd2 = open("alphabet.txt", O_RDONLY);
    int flag1 = 1, flag2 = 1;
    while ((flag1 == 1) && (flag2 == 1))
    {
        if (1 == (flag1 = read(fd1, &c, 1)))
        {
            write(1, &c, 1);
            if (1 == (flag2 = read(fd2, &c, 1)))
                write(1, &c, 1);
        }
    }
    return 0;
}
```



```
^^b/s/o/l/l/src >>> ./2.out
AAbbccddeeffgghhiijjkkllmmnnnooppqrrssttuuvvwwxxyyzz
```

Рисунок 2.3 — Результат выполнения программы

Во второй программе один и тот же файл открывается два раза только для чтения (`O_RDONLY`). Системный вызов `open()` создает дескриптор открытого файла в таблице открытых файлов процесса и запись в системной таблице открытых файлов. Так как файл открывается дважды, то в системной таблице создается два дескриптора `struct file`, каждый из которых имеет собственный указатель `f_pos`. Таким образом, в данном случае чтение из файла является независимым: при поочередном вызове `read()` для каждого дескриптора соответствующие указатели `f_pos` проходят по всем позициям файла, каждый символ считывается и выводится по два раза.

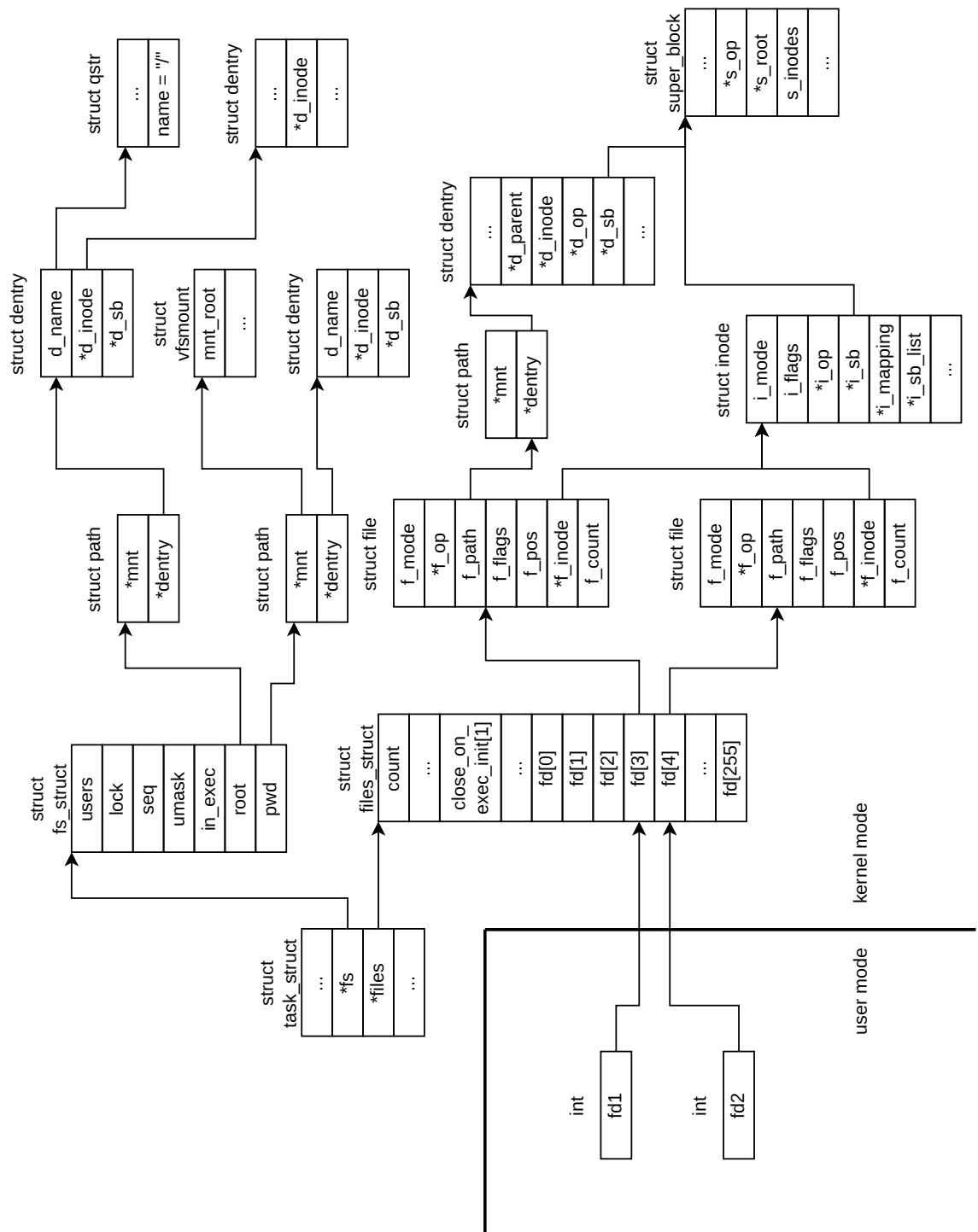
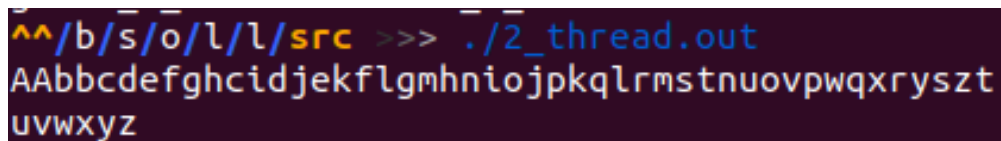


Рисунок 2.4 — Связи структур во второй программе

2.2.2 С использованием двух дополнительных потоков

Листинг 2.3 – Вторая программа, первый вариант (два дополнительных потока)

```
#include <stdio.h>
#include <pthread.h>
#include <fcntl.h>
#include <unistd.h>
void *thread_start(void *arg)
{
    int *fd = arg;
    char c;
    while (read(*fd, &c, 1))
        write(1, &c, 1);
    return NULL;
}
int main()
{
    int fd[2] = {open("alphabet.txt", O_RDONLY),
                 open("alphabet.txt", O_RDONLY)};
    pthread_t thr[2];
    for (int i = 0; i < 2; i++)
        if (pthread_create(&thr[i], NULL, thread_start, &fd[i]))
        {
            perror("pthread_create");
            return 1;
        }
    close(fd[0]);
    close(fd[1]);
    return 0;
}
```

A terminal window with a dark purple background. The prompt is '^/b/s/o/l/l/src >>>'. The command executed is './2_thread.out'. The output consists of two lines of lowercase letters: 'AAbbcdefghcidjekflgmhniojpkqlrmstnuovpwqxrysz' on the first line and 'tuvwxyz' on the second line. The letters are colored in a rainbow gradient.

```
^/b/s/o/l/l/src >>> ./2_thread.out
AAbbcdefghcidjekflgmhniojpkqlrmstnuovpwqxrysz
tuvwxyz
```

Рисунок 2.5 — Результат выполнения программы

В многопоточной версии программы порядок вывода символов не определен, так как потоки выполняются параллельно (асинхронно). В случае с главным и дополнительным потоками дополнительный поток начинает выполнение позже главного, так как на его создание затрачивается время.

2.3 Вторая программа, второй вариант

2.3.1 Без использования потоков

Листинг 2.4 – Вторая программа, второй 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 = %ld, size = %ld bytes\n", \
            statbuf.st_ino, statbuf.st_size); \
    } while (0);
int main()
{
    int fd1 = open("q.txt", O_RDWR);
    PRINT_STAT("open fd1 ");
    int fd2 = open("q.txt", O_RDWR);
    PRINT_STAT("open fd2 ");
    for (char c = 'a'; c <= 'z'; c++)
    {
        if (c % 2)
            write(fd1, &c, 1);
        else
            write(fd2, &c, 1);
        PRINT_STAT("write      ");
    }
    close(fd1);
    PRINT_STAT("close fd1");
    close(fd2);
    PRINT_STAT("close fd2");
    return 0;
}
```

```

^/b/s/o/l/l/src >>> ./2_1.out
open fd1 : inode = 796009, size = 1 bytes
open fd2 : inode = 796009, size = 1 bytes
write    : inode = 796009, size = 1 bytes
write    : inode = 796009, size = 1 bytes
write    : inode = 796009, size = 2 bytes
write    : inode = 796009, size = 2 bytes
write    : inode = 796009, size = 3 bytes
write    : inode = 796009, size = 3 bytes
write    : inode = 796009, size = 4 bytes
write    : inode = 796009, size = 4 bytes
write    : inode = 796009, size = 5 bytes
write    : inode = 796009, size = 5 bytes
write    : inode = 796009, size = 6 bytes
write    : inode = 796009, size = 6 bytes
write    : inode = 796009, size = 7 bytes
write    : inode = 796009, size = 7 bytes
write    : inode = 796009, size = 8 bytes
write    : inode = 796009, size = 8 bytes
write    : inode = 796009, size = 9 bytes
write    : inode = 796009, size = 9 bytes
write    : inode = 796009, size = 10 bytes
write    : inode = 796009, size = 10 bytes
write    : inode = 796009, size = 11 bytes
write    : inode = 796009, size = 11 bytes
write    : inode = 796009, size = 12 bytes
write    : inode = 796009, size = 12 bytes
write    : inode = 796009, size = 13 bytes
write    : inode = 796009, size = 13 bytes
close fd1: inode = 796009, size = 13 bytes
close fd2: inode = 796009, size = 13 bytes
^/b/s/o/l/l/src >>> cat q.txt
bdfhjlnprtvxz

```

Рисунок 2.6 — Результат выполнения программы

В программе один и тот же файл ("q.txt") открывается дважды для чтения и записи (O_RDWR). Системный вызов `open()` создает дескриптор открытого файла в таблице открытых файлов процесса и запись в системной таблице открытых файлов. Так как файл открывается дважды, то в системной таблице создается два дескриптора `struct file`, каждый из которых имеет собственный указатель `f_pos`. При первом вызове `write()` для `fd1` символ 'а' записывается в файл на 0 позицию и соответствующий указатель `f_pos` увеличивается на 1. При первом вызове `write()` для `fd2` символ 'b' также записывается в файл на 0 позицию и соответствующий указатель `f_pos` увеличивается на 1. Таким образом, при поочередной записи символов в файл он будет содержать только символы, которые записывались через `fd2`. Произошла потеря данных.

Чтобы избежать потерю данных, файл можно дважды открыть для добавления записи в конец (`O_RDWR | O_APPEND`). При первом вызове `write()` для `fd1` символ 'a' записывается в файл на последнюю позицию (0). При первом вызове `write()` для `fd2` символ 'b' также записывается в файл на последнюю позицию (1). Таким образом, при поочередной записи символов в файл он будет содержать все символы алфавита.

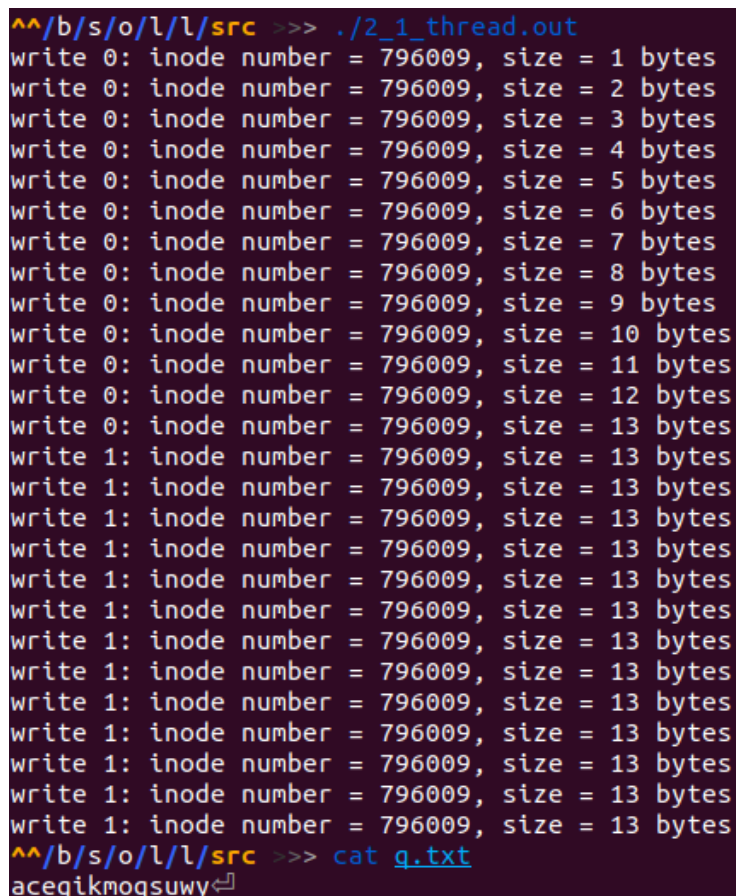
2.3.2 С использованием двух дополнительных потоков

Листинг 2.5 – Вторая программа, второй вариант (два дополнительных потока) (начало)

```
#include <fcntl.h>
#include <unistd.h>
#include <pthread.h>
#include <stdio.h>
#include <sys/stat.h>
struct stat statbuf;
#define PRINT_STAT(action, i) \
    do { \
        stat("q.txt", &statbuf); \
        fprintf(stdout, action " %d: inode number = %ld, size = %ld \
            bytes\n", \
            i, statbuf.st_ino, statbuf.st_size); \
    } while (0);
struct thread_arg {
    int fd;
    int i;
};
void *thread_start(void *arg)
{
    struct thread_arg *targ = arg;
    for (char c = 'a'; c <= 'z'; c++)
        if (c % 2 == targ->i)
        {
            write(targ->fd, &c, 1);
            PRINT_STAT("write", targ->i);
        }
    return NULL;
}
int main()
{
    int fd[2] = {open("q.txt", O_RDWR),
                 open("q.txt", O_RDWR)};
    pthread_t thr[2];
    struct thread_arg targ[2];
```

Листинг 2.6 – Вторая программа, второй вариант (два дополнительных потока) (конец)

```
for (int i = 0; i < 2; i++)
{
    targ[i].fd = fd[i];
    targ[i].i = i;
    if (pthread_create(&thr[i], NULL, thread_start, &targ[i]))
    {
        perror("pthread_create");
        return 1;
    }
}
close(fd[0]);
close(fd[1]);
return 0;
}
```



```
^/b/s/o/l/l/src >>> ./2_1_thread.out
write 0: inode number = 796009, size = 1 bytes
write 0: inode number = 796009, size = 2 bytes
write 0: inode number = 796009, size = 3 bytes
write 0: inode number = 796009, size = 4 bytes
write 0: inode number = 796009, size = 5 bytes
write 0: inode number = 796009, size = 6 bytes
write 0: inode number = 796009, size = 7 bytes
write 0: inode number = 796009, size = 8 bytes
write 0: inode number = 796009, size = 9 bytes
write 0: inode number = 796009, size = 10 bytes
write 0: inode number = 796009, size = 11 bytes
write 0: inode number = 796009, size = 12 bytes
write 0: inode number = 796009, size = 13 bytes
write 1: inode number = 796009, size = 13 bytes
write 1: inode number = 796009, size = 13 bytes
write 1: inode number = 796009, size = 13 bytes
write 1: inode number = 796009, size = 13 bytes
write 1: inode number = 796009, size = 13 bytes
write 1: inode number = 796009, size = 13 bytes
write 1: inode number = 796009, size = 13 bytes
write 1: inode number = 796009, size = 13 bytes
write 1: inode number = 796009, size = 13 bytes
write 1: inode number = 796009, size = 13 bytes
^/b/s/o/l/l/src >>> cat q.txt
acegikmoqsuwy
```

Рисунок 2.7 — Результат выполнения программы

В многопоточной версии программы потоки выполняются параллельно (асинхронно). Однако потеря данных полностью аналогична той, что однопоточной версии программы: при первом вызове `write()` для `fd[0]` (первый поток) символ 'a' записывается в файл на 0 позицию и соответствующий указатель `f_pos` увеличивается на 1; при первом вызове `write()` для `fd[1]` (второй поток) символ 'b' также записывается в файл на 0 позицию и соответствующий указатель `f_pos` увеличивается на 1. Таким образом, при поочередной записи символов в файл он будет содержать только символы, которые записывались вторым потоком.

Чтобы избежать потерю данных, файл можно дважды открыть для добавления записи в конец (`O_RDWR | O_APPEND`). При первом вызове `write()` для `fd[0]` (первый поток) символ 'a' записывается в файл на последнюю позицию (0). При первом вызове `write()` для `fd[1]` (второй поток) символ 'b' также записывается в файл на последнюю позицию (1). Таким образом, при поочередной записи символов в файл он будет содержать все символы алфавита. Порядок символов не определен, так как потоки выполняются параллельно (асинхронно).

2.4 Третья программа

Листинг 2.7 – Третья программа

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#define PRINT_STAT(action) \
    do { \
        stat("q.txt", &statbuf); \
        fprintf(stdout, action ": inode = %ld, size = %ld bytes\n", \
            statbuf.st_ino, statbuf.st_size); \
    } while (0);
struct stat statbuf;
int main()
{
    FILE *fs1 = fopen("q.txt", "w");
    PRINT_STAT("open fs1");
    FILE *fs2 = fopen("q.txt", "w");
    PRINT_STAT("open fs2");
    for (char c = 'a'; c <= 'z'; c++)
    {
        if (c % 2)
            fprintf(fs1, "%c", c);
        else
            fprintf(fs2, "%c", c);
        PRINT_STAT("fprintf");
    }
    fclose(fs1);
    PRINT_STAT("fclose fs1");
    fclose(fs2);
    PRINT_STAT("fclose fs2");
    return 0;
}
```

В третьей программе файл дважды открывается на чтение и запись функцией `fopen()` из библиотеки буферизованного ввода-вывода `stdio.h`. В результате выполнения `fopen()` в системной таблице открытых файлов создаются два дескриптора `struct file`, каждый из которых имеет собственный указатель `f_pos`, при этом оба дескриптора ссылаются на один и тот же `inode`. Библиотечная функция `fprintf()` выполняет буферизованный вывод. При этом

```
^~/b/s/o/l/l/src >>> ./3.out
open fs1      : inode = 796009, size = 0 bytes
open fs2      : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fprintf       : inode = 796009, size = 0 bytes
fclose fs1    : inode = 796009, size = 13 bytes
fclose fs2    : inode = 796009, size = 13 bytes
^~/b/s/o/l/l/src >>> cat q.txt
bdfhjlnprtvxz
```

Рисунок 2.8 — Результат выполнения программы

буфер создается неявно. Данные из буфера записываются в файл по трем причинам:

- 1) буфер заполнен,
- 2) вызвана функция `flush()` (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 и буфер для `fs2` записывается в файл. Таким образом, произошла потеря данных, так как в файле находится только содержимое буфера для `fs2`.

Чтобы избежать потерю данных, при втором открытии файла можно указать режим записи в конец файла ("a"). В таком случае содержимое при вызове `fclose()` для `fs2` содержимое файла не будет очищаться, а содержимое буфера для `fs2` будет добавлено в конец файла.

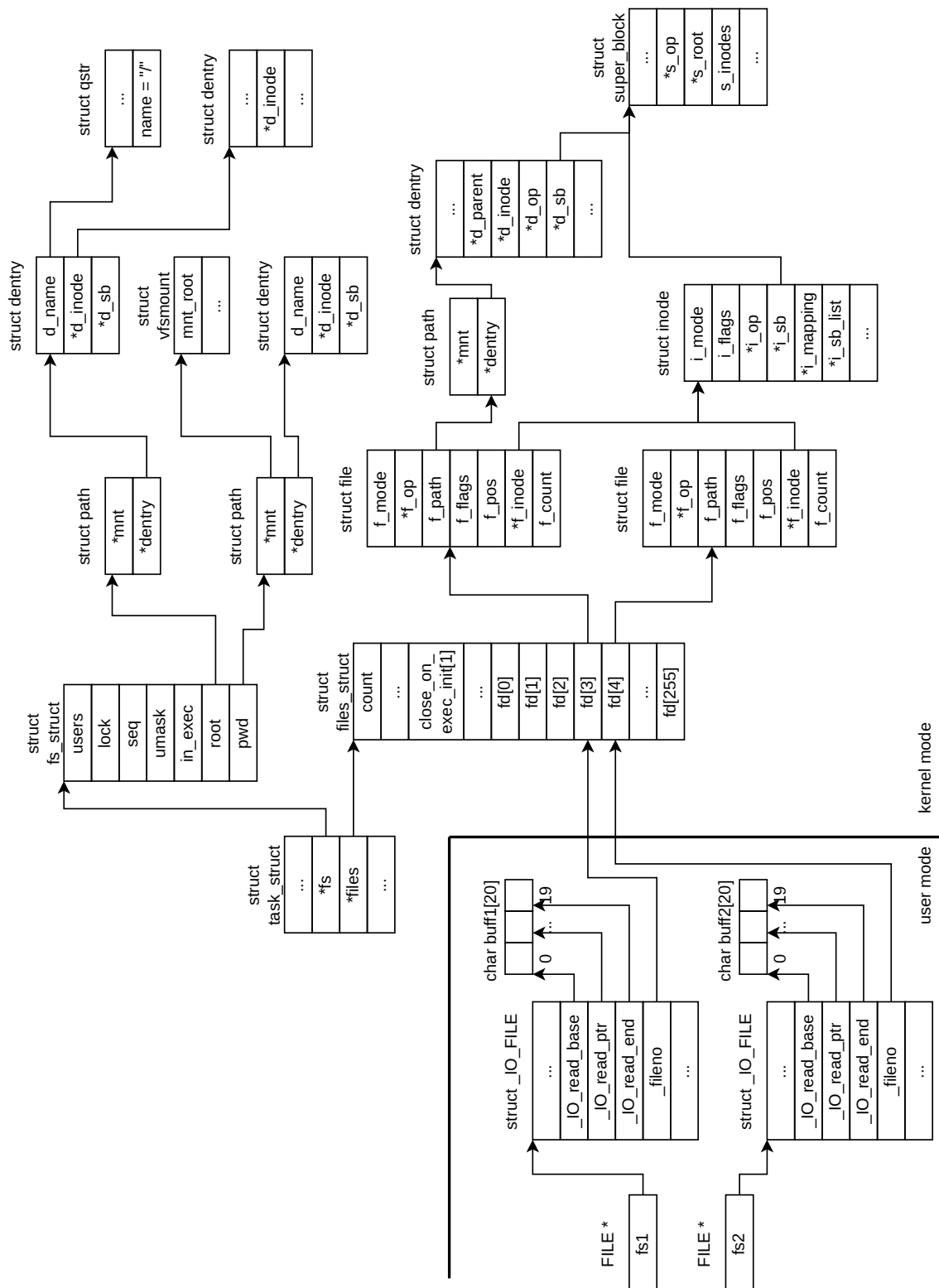


Рисунок 2.9 — Связи структур в третьей программе