



**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»

Лабораторная работа
по дисциплине «Операционные системы»

Тема Буферизованный и небуферизованный ввод-вывод

Студент Пермякова Е. Д.

Группа ИУ7-62Б

Преподаватель Рязанова Н.Ю

Москва, 2025

1 Структура FILE

Листинг 1.1 – Структура FILE

```
/* /usr/include/x86_64-linux-gnu/bits/types/FILE.h */
#ifndef __FILE_defined
#define __FILE_defined 1

struct _IO_FILE;

/* The opaque type of streams. This is the definition used
   elsewhere. */
typedef struct _IO_FILE FILE;

#endif

/* /usr/include/x86_64-linux-gnu/bits/types/struct_FILE.h */
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.  */

/* 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
};

```

2 Первая программа

2.1 Код

Листинг 2.1 – Программа 1

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

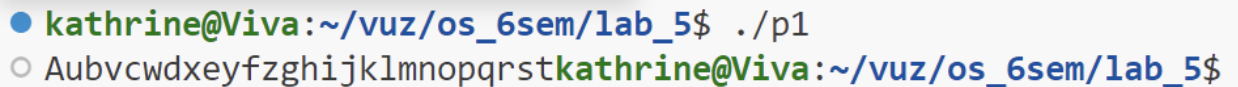
int main()
{
    // have kernel open connection to file alphabet.txt
    int fd = open("alphabet.txt", O_RDONLY);

    // create two a C I/O buffered streams using the above
    connection
    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);

    // read a char & write it alternately from fs1 and fs2
    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;
}
```

2.2 Результат работы программы



```
● kathrine@Viva:~/vuz/os_6sem/lab_5$ ./p1
○ Aubvcwdxeyfzghijklmnopqrstkathrine@Viva:~/vuz/os_6sem/lab_5$
```

Рисунок 2.1 – Результат работы программы 1

2.3 Анализ полученного результата

Вызывается системный вызов `open`, который открывает файл "alphabet.txt" для чтения и возвращает дескриптор в таблице открытых файлов процесса (`fd`).

Два раза вызывается функция `fdopen`, которой передается полученный файловый дескриптор `fd`. В результате инициализируются две структуры `_IO_FILE` (`fs1`, `fs2`) библиотеки буферизованного ввода-вывода `stdio`. Вызывается функция `setvbuf` с флагом `_IOFBF`, который определяется тип буферизации – полный.

В начале цикла вызывается функция `fscanf`, которой передается `fs1`, в буфер структуры `fs1` записываются первые 20 символов ("Abcdefghijklmnopqrst"), смещается указатель на 20, функции `fprintf` передается символ 'A'. Вызывается функция `fscanf`, которой передается `fs2`, в буфер структуры `fs2` записываются оставшиеся символы ("vwxyz"), смещается указатель на 6, функции `fprintf` передается символ 'u'.

В результате символы из буферов будут поочередно выведены на экран.

2.4 Связь структур файловой подсистемы

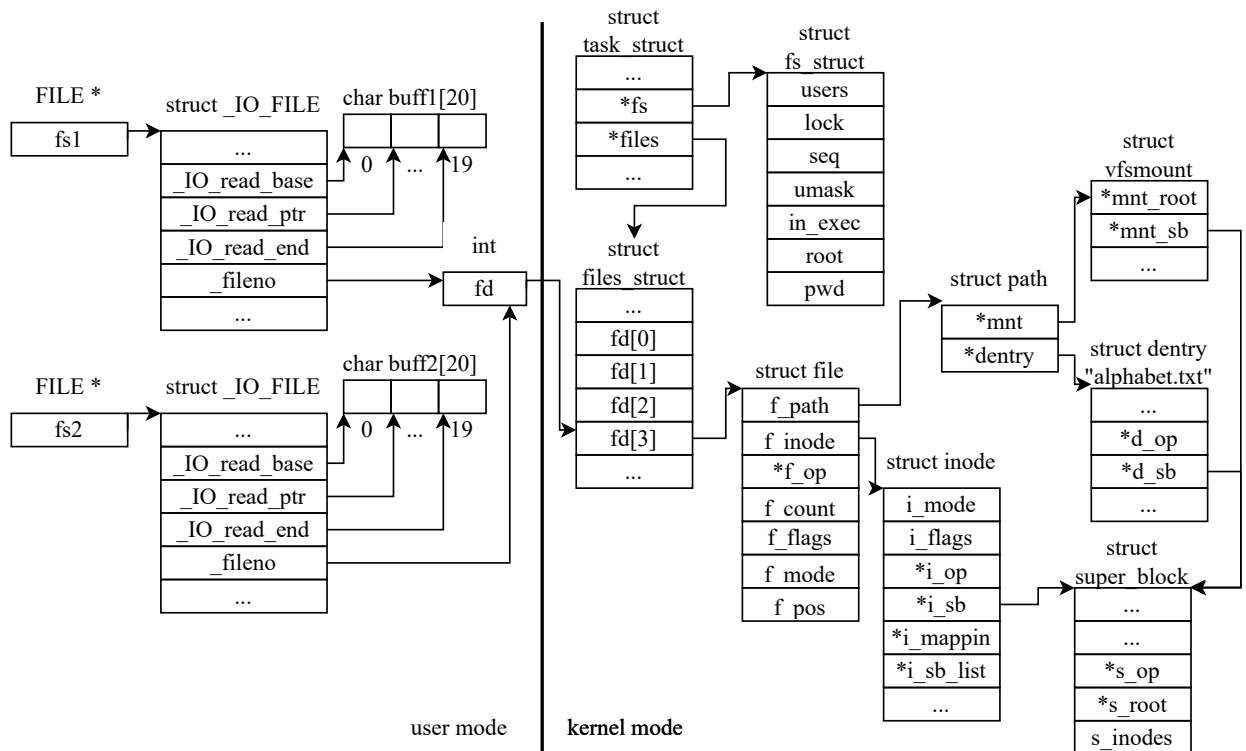


Рисунок 2.2 – Связь структур файловой подсистемы

3 Вторая программа, вариант 1

3.1 Однопоточная версия


3.1.1 Код

Листинг 3.1 – Программа 2, вариант 1, однопоточной версия

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

int main()
{
    char c;
    // have kernel open two connection to file alphabet.txt
    int fd1 = open("alphabet.txt", O_RDONLY);
    int fd2 = open("alphabet.txt", O_RDONLY);
    int fl1 = 1, fl2 = 1;
    while (fl1 || fl2) {
        fl1 = read(fd1, &c, 1);
        if (fl1 == 1)
            write(1, &c, 1);
        fl2 = read(fd2, &c, 1);
        if (fl2 == 1)
            write(1, &c, 1);
    }
    return 0;
}
```

3.1.2 Результат работы программы



```
kathrine@Viva:~/vuz/os_6sem/lab_5$ ./p21
AAbbccddeeffgghhiijjkkllmmnnooppqrrssttuuvvwwxxyyzz
```

Рисунок 3.1 – Результат работы однопоточной версии программы 2, вариант 1

3.1.3 Анализ полученного результата

Два раза вызывается системный вызов `open`, который открывает файл "alphabet.txt" для чтения и возвращает дескрипторы в таблице открытых файлов

процесса (fd1 и fd2). Оба дескриптора ссылаются на один и тот же объект inode, но создаются 2 записи в таблице дескрипторов открытых файлов процесса объекта files_struct. Так как объекты struct file соответствующие дескрипторами fd1 и fd2 разные, они имеют разные поля f_pos.

Вызывается системный вызов read с параметром fd1 и из файла считывается символ 'A', указатель f_pos объекта struct file, соответствующий fd1, смещается на 1. Символ выводится на экран. Вызывается системный вызов read с параметром fd2. Указатель f_pos, соответствующий fd2, равен 0 и из файла считывается символ 'A', этот указатель смещается на 1. Символ выводится на экран.

В результате каждая буква алфавита будет выведена 2 раза, так как использовалось 2 разных объекта struct file, каждый со своим значением поля f_pos.

3.2 Многопоточная версия

3.2.1 Код

Листинг 3.2 – Программа 2, вариант 1, многопоточная версия

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

void *thread_func(void *arg)
{
    int fd = *(int *)arg;
    char c;
    int fl = 1;
    while (fl) {
        fl = read(fd, &c, 1);
        if (fl == 1)
            write(1, &c, 1);
    }
    return NULL;
}

int main()
```



```

{
    int fd1 = open("alphabet.txt", O_RDONLY);
    int fd2 = open("alphabet.txt", O_RDONLY);
    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
    pthread_t t;
    if (pthread_create(&t, &attr, thread_func, &fd1)) {
        perror("pthread_create");
        exit(1);
    }
    char c;
    int fl = 1;
    while (fl) {
        fl = read(fd2, &c, 1);
        if (fl == 1)
            write(1, &c, 1);
    }
    return 0;
}

```

3.2.2 Результат работы программы

```

kathrine@Viva:~/vuz/os_6sem/lab_5$ ./p22
AbcdefgAhbicjdkelfmgnhoipjqkrslmtunvowpxqyrzstkathrine@Viva:~/vuz/os_6sem/lab_5$ ./p22
AbcdAebfcgdheifjgkhlmjnkolpqnrosptqurvswtxuyvzwxkathrine@Viva:~/vuz/os_6sem/lab_5$

```

Рисунок 3.2 – Результат работы многопоточной версии программы 2, вариант

1

3.2.3 Анализ полученного результата

Два раза вызывается системный вызов `open`, который открывает файл "alphabet.txt" для чтения и возвращает дескрипторы в таблице открытых файлов процесса (`fd1` и `fd2`). Оба дескриптора ссылаются на один и тот же объект `inode`, но создаются 2 записи в таблице дескрипторов открытых файлов процесса объекта `files_struct`, у каждого объекта `files_struct` свое значение `f_pos`.

Создается отсоединенный поток, который выполняет то же самое что и основной, за исключением того, что отсоединенный поток работает с дескриптором `fd1`, а основной с `fd2`. В каждом потоке вызывается системный

вызов `read` и из соответствующего файла считывается символ, указатель `f_pos` объекта `struct file` смещается на 1. Символ выводится на экран. Так как на создание потока требуется время, то главный поток успевает прочитать и вывести часть символов. Далее потоки поочерёдно читают и выводят символы, при этом когда главный поток заканчивает чтение, процесс завершается и дополнительный поток не успевает дочитать до конца файла.

4 Вторая программа, вариант 2

4.1 Однопоточная версия

4.1.1 Код

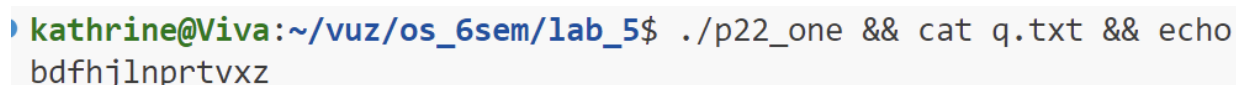
Листинг 4.1 – Программа 2, вариант 2, однопоточная версия

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

int main()
{
    int fd1 = open("q.txt", O_RDWR);
    int fd2 = open("q.txt", O_RDWR);

    for(char c = 'a'; c <= 'z'; c++) {
        if (c%2){
            write(fd1, &c, 1);
        } else {
            write(fd2, &c, 1);
        }
    }
    close(fd1);
    close(fd2);
    return 0;
}
```

4.1.2 Результат работы программы



```
kathrine@Viva:~/vuz/os_6sem/lab_5$ ./p22_one && cat q.txt && echo
bdfhjlnprtvxz
```

Рисунок 4.1 – Результат работы однопоточной версии программы 2, вариант 2

4.1.3 Анализ полученного результата

Два раза вызывается системный вызов `open`, который открывает файл "q.txt" для чтения и записи и возвращает дескрипторы в таблице открытых файлов процесса (`fd1` и `fd2`). Оба дескриптора ссылаются на один и тот же объект `inode`, но создаются 2 записи в таблице дескрипторов открытых файлов

процесса объекта `files_struct`, у каждого объекта `files_struct` свое значение `f_pos`.

В начале цикла системному вызову `write` передается `fd2`, так как `'a' = 97, 97 % 2 == 1`. В начало файла записывается символ `'a'`, `f_pos` объекта `struct file`, соответствующего дескриптору `fd2`, смещается на 1.

Во втором проходе цикла системному вызову `write` передается `fd1`. В начало файла записывается символ `'b'`, так как `f_pos` объекта `struct file`, соответствующего дескриптору `fd1` равен 0. `f_pos` смещается на 1.

В результате в файл будут записаны только символы, стоящие на чётных позициях в алфавите.

4.2 Многопоточная версия

4.2.1 Код

Листинг 4.2 – Программа 2, вариант 2, многопоточная версия

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

void *thread_func(void *arg)
{
    int fd = *(int *)arg;
    for(char c = 'a'; c <= 'z'; c++) {
        if (c % 2 == 0) {
            write(fd, &c, 1);
        }
    }
    return NULL;
}

int main()
{
    int fd1 = open("q.txt", O_RDWR);
    int fd2 = open("q.txt", O_RDWR);
    pthread_attr_t attr;
    pthread_attr_init(&attr);
```

```

pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
pthread_t t1;
if (pthread_create(&t1, &attr, thread_func, &fd1)) {
    perror("pthread_create");
    exit(1);
}
for(char c = 'a'; c <= 'z'; c++) {
    if (c%2 == 1) {
        write(fd2, &c, 1);
    }
}
close(fd1);
close(fd2);
return 0;
}

```

4.2.2 Результат работы программы

```

kathrine@Viva:~/vuz/os_6sem/lab_5$ ./p22_th && cat q.txt && echo
bcegi kmoqsuwy
kathrine@Viva:~/vuz/os_6sem/lab_5$ ./p22_th && cat q.txt && echo
bdfhjlnprtvxz

```

Рисунок 4.2 – Результат работы многопоточная версии программы 2, вариант 2

4.2.3 Анализ полученного результата

Два раза вызывается системный вызов `open`, который открывает файл "q.txt" для чтения и записи и возвращает дескрипторы в таблице открытых файлов процесса (`fd1` и `fd2`). Оба дескриптора ссылаются на один и тот же объект `inode`, но создаются 2 записи в таблице дескрипторов открытых файлов процесса объекта `files_struct`, у каждого объекта `files_struct` свое значение `f_pos`.

Создается отсоединенный поток, который выполняет то же самое что и основной, за исключением того, что отсоединенный поток работает с дескриптором `fd1`, а основной с `fd2`. Поток параллельно записывают в один файл и, так как они используют разные индексы открытых файлов, то один поток перезаписывает символы другого.

Таким образом в файл будут записаны символы, стоящие как на чётных так и на нечетных позициях в алфавите.

4.3 Связь структур файловой подсистемы

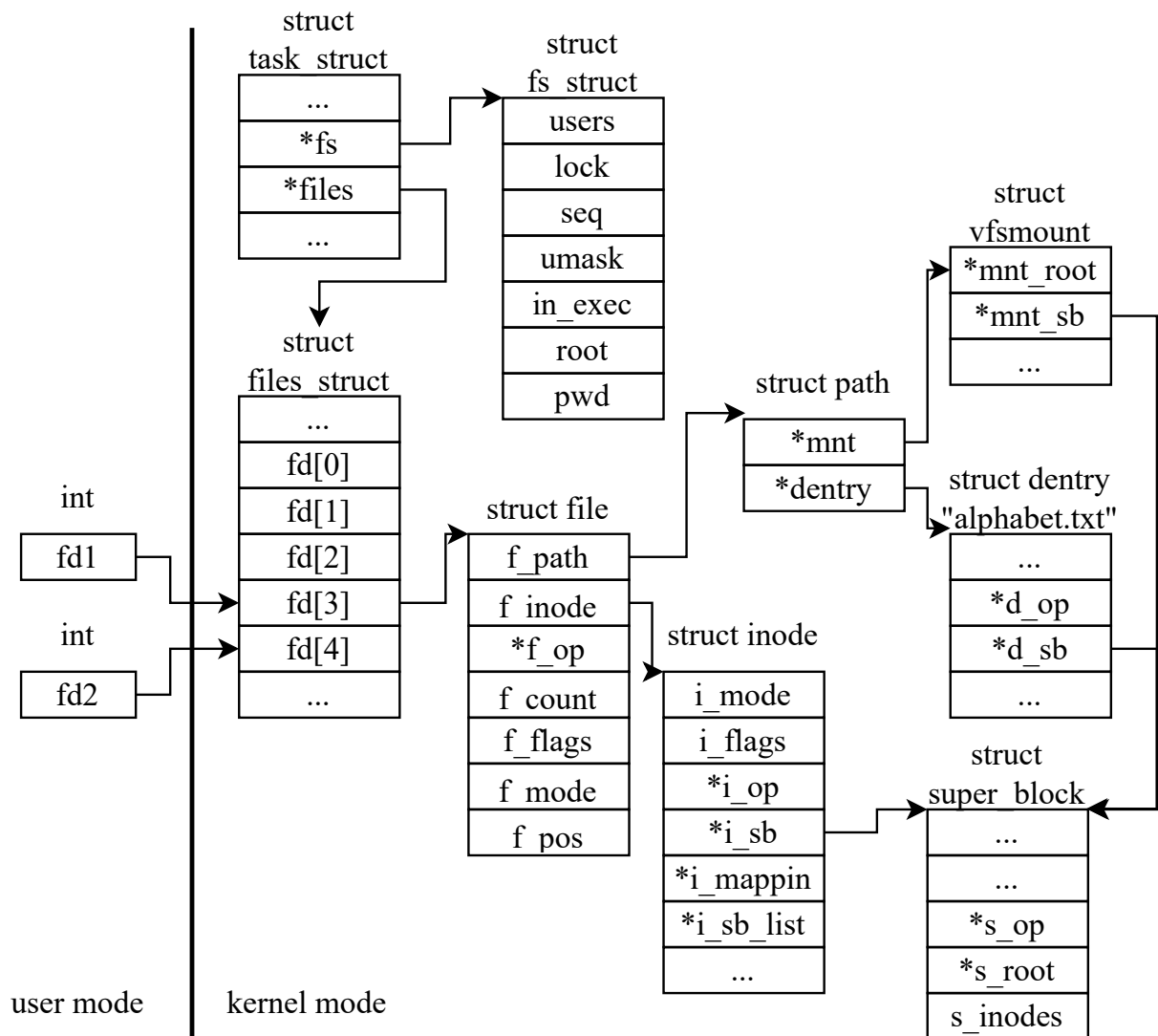


Рисунок 4.3 – Связь структур файловой подсистемы

5 Третья программа

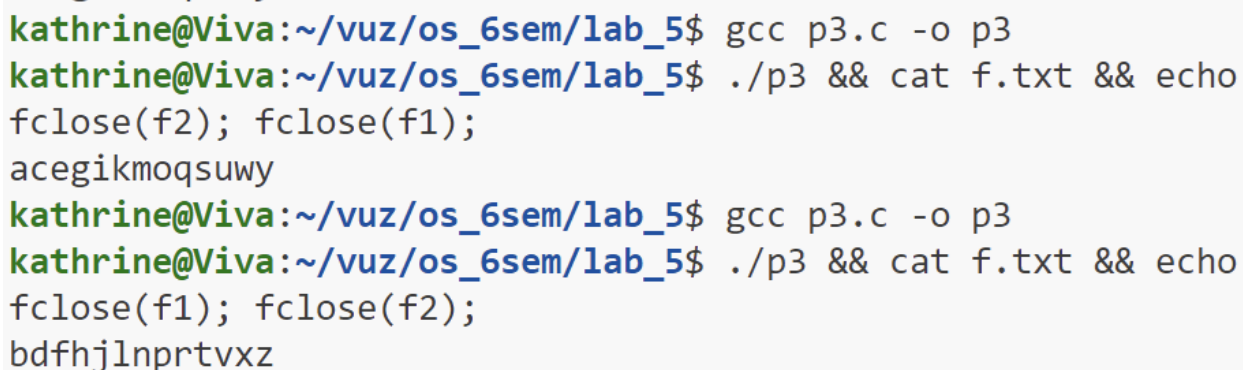
5.1 Код

Листинг 5.1 – Программа 3

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

int main()
{
    FILE *f1, *f2;
    f1 = fopen("f.txt", "w");
    f2 = fopen("f.txt", "w");
    for(char c = 'a'; c <= 'z'; c++) {
        if (c%2)
            fprintf(f1, "%c", c);
        else
            fprintf(f2, "%c", c);
    }
    printf("fclose(f1); fclose(f2);\n");
    fclose(f1);
    fclose(f2);
    return 0;
}
```

5.2 Результат работы программы



```
kathrine@Viva:~/vuz/os_6sem/lab_5$ gcc p3.c -o p3
kathrine@Viva:~/vuz/os_6sem/lab_5$ ./p3 && cat f.txt && echo
fclose(f2); fclose(f1);
acegikmoqsuwy
kathrine@Viva:~/vuz/os_6sem/lab_5$ gcc p3.c -o p3
kathrine@Viva:~/vuz/os_6sem/lab_5$ ./p3 && cat f.txt && echo
fclose(f1); fclose(f2);
bdfhjlnprtvxz
```

Рисунок 5.1 – Результат работы программы 3

5.3 Анализ полученного результата

Два раза вызывается функция `open`, которая открывает файл "f.txt" и инициализирует 2 структуры `_IO_FILE` (`f1` и `f2`). В цикле в буфер структуры `f1` записываются символы, стоящие на чётных позициях в алфавите, а в буфер структуры `f2` – на нечетных.

Так как используется буферизация, то содержимое, записанное в буфер объекта `_IO_FILE` будет записано в файл в одном из следующих случаев:

- буфер заполнен;
- вызов функции `fflush()`;
- вызов функции `fclose()`;

В результате в программе содержимое файла определяется порядком вызова `fclose()`: если первым вызвать `fclose` для `f1`, то при вызове `fclose()` для `f2` содержимое файла будет перезаписано содержимым буфера `f2` и наоборот.

Чтобы избежать потери данных, файл следует открывать в режиме добавления - `O_APPEND`, в таком случае запись каждый раз будет производиться в реальный конец файла.

5.4 Связь структур файловой подсистемы

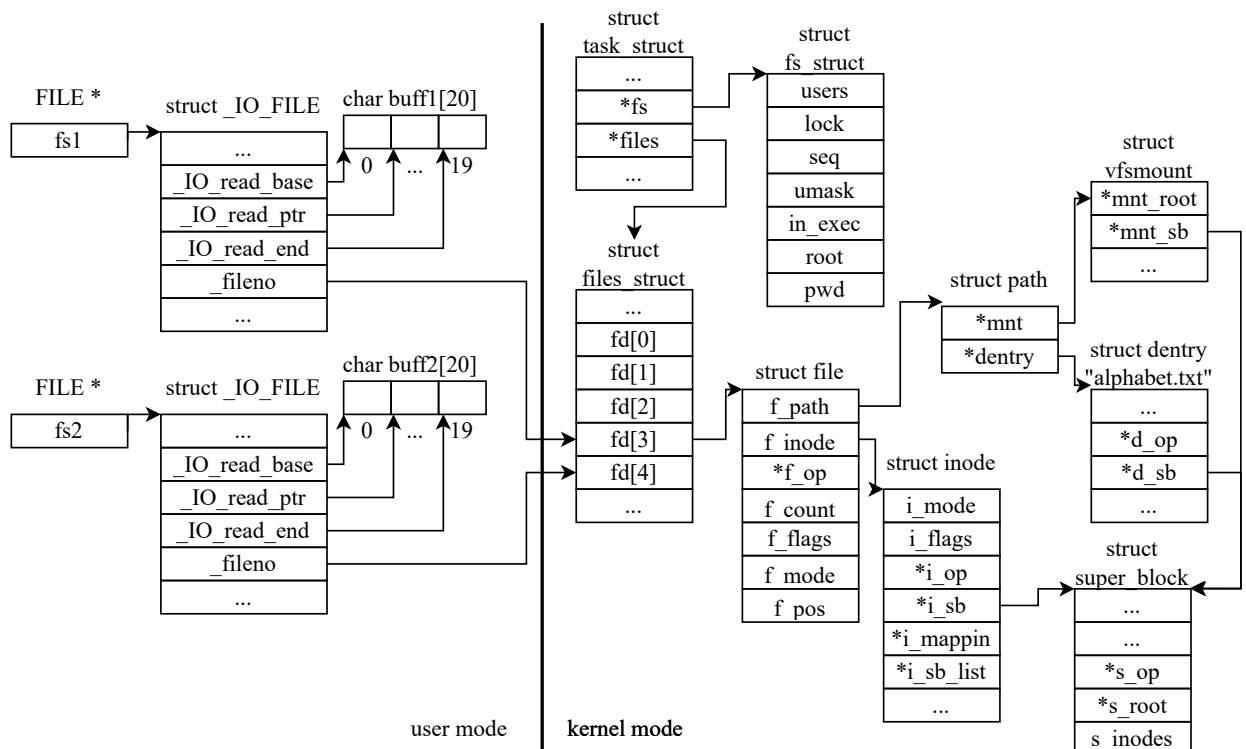


Рисунок 5.2 – Связь структур файловой подсистемы