



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

НАЗВАНИЕ \_\_\_\_\_ Буферизованный и небуферизованный ввод-вывод

ДИСЦИПЛИНА \_\_\_\_\_ Операционные системы

Студент ИУ7-64Б  
(Группа)

\_\_\_\_\_  
(Подпись, дата) Я. Р. Цховребова  
(И. О. Фамилия)

Преподаватель

\_\_\_\_\_  
(Подпись, дата) Н. Ю. Рязанова  
(И. О. Фамилия)

2025 г.

# 1 Структура FILE

```
1 typedef struct __sFILE {
2     unsigned char *_p; /* current position in (some) buffer */
3     int _r; /* read space left for getc() */
4     int _w; /* write space left for putc() */
5     short _flags; /* flags, below; this FILE is free if 0 */
6     short _file; /* fileno, if Unix descriptor, else -1 */
7     struct __sbuf _bf; /* the buff (at least 1 byte, if !NULL) */
8     int _lbfsz; /* 0 or _bf._size, for inline putc */
9     /* operations */
10    void *_cookie; /* cookie passed to io functions */
11    int (* __Nullable _close)(void *);
12    int (* __Nullable _read)(void *, char *, int);
13    fpos_t (* __Nullable _seek)(void *, fpos_t, int);
14    int (* __Nullable _write)(void *, const char *, int);
15    /* separate buffer for long sequences of ungetc() */
16    struct __sbuf _ub; /* ungetc buffer */
17    struct __sFILEX *_extra; /* additions to FILE to not break
18        ABI */
19    int _ur; /* saved _r when _r is counting ungetc data */
20    /* tricks to meet minimum requirements even when malloc()
21        fails */
22    unsigned char _ubuf[3]; /* guarantee an ungetc() buffer */
23    unsigned char _nbuf[1]; /* guarantee a getc() buffer */
24    /* separate buffer for fgetln() when line crosses buffer
25        boundary */
26    struct __sbuf _lb; /* buffer for fgetln() */
27    /* Unix stdio files get aligned to block boundaries on
28        fseek() */
29    int _blksize; /* stat.st_blksize (may be != _bf._size) */
30    fpos_t _offset; /* current lseek offset (see WARNING) */
31 } FILE;
```

Листинг 1 – Описание структуры FILE (\_\_sFILE)

## Сравнение полей реализаций структуры FILE (BSD и glibc)

Обе структуры реализуют структуру типа FILE, определенной в файле стандартных описаний «stdio.h».

Наиболее распространены две реализации:

- `__sFILE` — используется в BSD-системах (FreeBSD, macOS),
- `_IO_FILE` — используется в GNU Libc (Linux).

Назначение	BSD ( <code>__sFILE</code> )	GNU ( <code>_IO_FILE</code> )
Указатель на текущую позицию в буфере	<code>unsigned char *_p</code>	<code>char *_IO_read_ptr</code> , <code>char *_IO_write_ptr</code>
Доступные байты для чтения	<code>int _r</code> (read count)	Разница между <code>_IO_read_end</code> и <code>_IO_read_ptr</code> (оба типа <code>char *</code> )
Доступные байты для записи	<code>int _w</code> (write count)	Разница между <code>_IO_write_end</code> и <code>_IO_write_ptr</code> (оба типа <code>char *</code> )
Буфер	<code>struct __sbuf _bf</code> (Поля: <code>unsigned char *_base</code> ; <code>int _size</code> ;) )	<code>char *_IO_buf_base</code> , <code>char *_IO_buf_end</code>
Номер дескриптора	<code>short _file</code>	<code>int _fileno</code>
Размер системного блока	<code>int _blksize</code>	<code>int _blksize</code>
Смещение в файле	<code>fpos_t _offset</code>	<code>off64_t _old_offset</code>
Флаги	<code>short _flags</code>	<code>int _flags</code> , <code>int _flags2</code>
Минимальные буферы	<code>unsigned char _ubuf[3]</code> , <code>unsigned char _nbuf[1]</code>	<code>char _shortbuf[1]</code> , <code>int _cur_column</code>

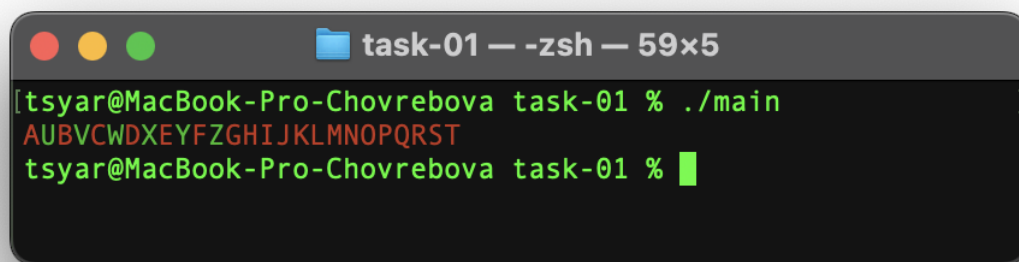
Механизм блокировки	Отсутствует явно	void *_lock — указатель на мьютекс
Системные функции ввода-вывода	<p>Указатели на функции:</p> <ul style="list-style-type: none"> <li>— int (*_read)(void *, char *, int);</li> <li>— int (*_write)(void *, const char *, int);</li> <li>— fpos_t (*_seek)(void *, fpos_t, int);</li> <li>— int (*_close)(void *);</li> </ul> <p>Контекст: void *_cookie</p>	<p>Вызов через vtable:</p> <pre>struct _IO_jump_t *_vtable  — ssize_t (*_IO_read)(...)  — ssize_t (*_IO_write)(...)  — off64_t (*_IO_seekoff)(...)  — и др.</pre>
Поддержка ungetc() — откат чтения	<pre>struct __sbuf _ub; int _ur</pre>	<pre>char *_IO_backup_base, _IO_save_base, _IO_save_end</pre>
Поддержка fgetln()	<pre>struct __sbuf _lb</pre>	Нет аналога
Расширения	<pre>void *_extra</pre>	<pre>struct _IO_marker *_markers, struct _IO_FILE *_chain</pre>

## 2 Программа №1

### Однопоточная реализация

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <fcntl.h>
4 #define FILE_NAME    "../alphabet.txt"
5 #define BUFF_SIZE    20
6 #define RED          "\x1b[31m"
7 #define GREEN        "\x1b[32m"
8 #define RESET        "\x1b[0m"
9
10 int main() {
11     int fd = open(FILE_NAME, O_RDONLY);
12     if (fd == -1) {
13         perror("open");
14         exit(EXIT_FAILURE);
15     }
16     FILE *fs1 = fdopen(fd, "r");
17     char buf1[BUFF_SIZE];
18     setvbuf(fs1, buf1, _IOFBF, BUFF_SIZE);
19     FILE *fs2 = fdopen(fd, "r");
20     char buf2[BUFF_SIZE];
21     setvbuf(fs2, buf2, _IOFBF, BUFF_SIZE);
22     int rc1 = 1, rc2 = 1;
23     while (rc1 == 1 || rc2 == 1) {
24         char c;
25         rc1 = fscanf(fs1, "%c", &c);
26         if (rc1 == 1)
27             fprintf(stdout, RED "%c" RESET, c);
28         rc2 = fscanf(fs2, "%c", &c);
29         if (rc2 == 1)
30             fprintf(stdout, GREEN "%c" RESET, c);
31     }
32     fclose(fs1);
33     fclose(fs2);
34     exit(EXIT_SUCCESS);
35 }
```

Листинг 2 – Первая программа (однопоточная)

A terminal window titled "task-01 — -zsh — 59x5" on a Mac. The prompt is "tsyar@MacBook-Pro-Chovrebova task-01 %". The user enters "./main" and the program outputs the alphabet "AUBVCWDXEYFZGHIJKLMNOPQRST" in a rainbow color scheme. The prompt returns to "tsyar@MacBook-Pro-Chovrebova task-01 %".

```
task-01 — -zsh — 59x5
[tsyar@MacBook-Pro-Chovrebova task-01 % ./main
AUBVCWDXEYFZGHIJKLMNOPQRST
tsyar@MacBook-Pro-Chovrebova task-01 % ]
```

Рисунок 1 – Результат работы программы первой однопоточной программы

## Анализ работы программы

Программа считывает информацию из файла «alphabet.txt», который содержит английский алфавит – строку символов. В результате своей работы программа при помощи двух буферов посимвольно выводит считанные символы в стандартный поток вывода stdout.

Во время выполнения программы вызывается системный вызов **open()**, который создает новый файловый дескриптор для открытого в режиме «только для чтения» файла «alphabet.txt» и возвращает созданный файловый дескриптор из системной таблицы открытых процессом файлов.

Дважды вызывается функция **fdopen()**, которая связывает поток ввода-вывода со существующим файловым дескриптором. Каждый вызов **fdopen()** создает отдельную структуру `__sFILE` (BSD) либо `_IO_FILE` (Linux, GNU), связанную с одним и тем же файловым дескриптором, и возвращает указатель типа `FILE` на эту структуру. Таким образом, получаются два независимых потока, использующих один и тот же дескриптор файла, но имеющих собственные буферы.

Далее с помощью функции **setvbuf()** изменяется тип буферизации для **fs1** и **fs2** на полную буферизацию – при полной буферизации данные хранятся в буфере и записываются в файл только тогда, когда буфер полностью заполнен; также явно задается размер буфера **BUFF\_SIZE** равный 20.

В цикле происходит считывание символов из файла с помощью функции **fscanf()**. При первом вызове **fscanf()** данные записываются в буфер

**buf1**: первые 20 символов из файла, а указатель текущей позиции в файле (**f\_pos**) перемещается на позицию, соответствующую следующему байту после последнего считанного. При следующем вызове **fscanf()** в буфер **buf2** записываются оставшиеся символы файла, после чего указатель **f\_pos** перемещается в конец файла.

При выводе содержимого буферов **buf1** и **buf2** с помощью вызова **fprintf()** символы считываются поочерёдно из двух буферов. В результате после символа 'А' из первого буфера сразу следует символ 'U' из второго буфера.

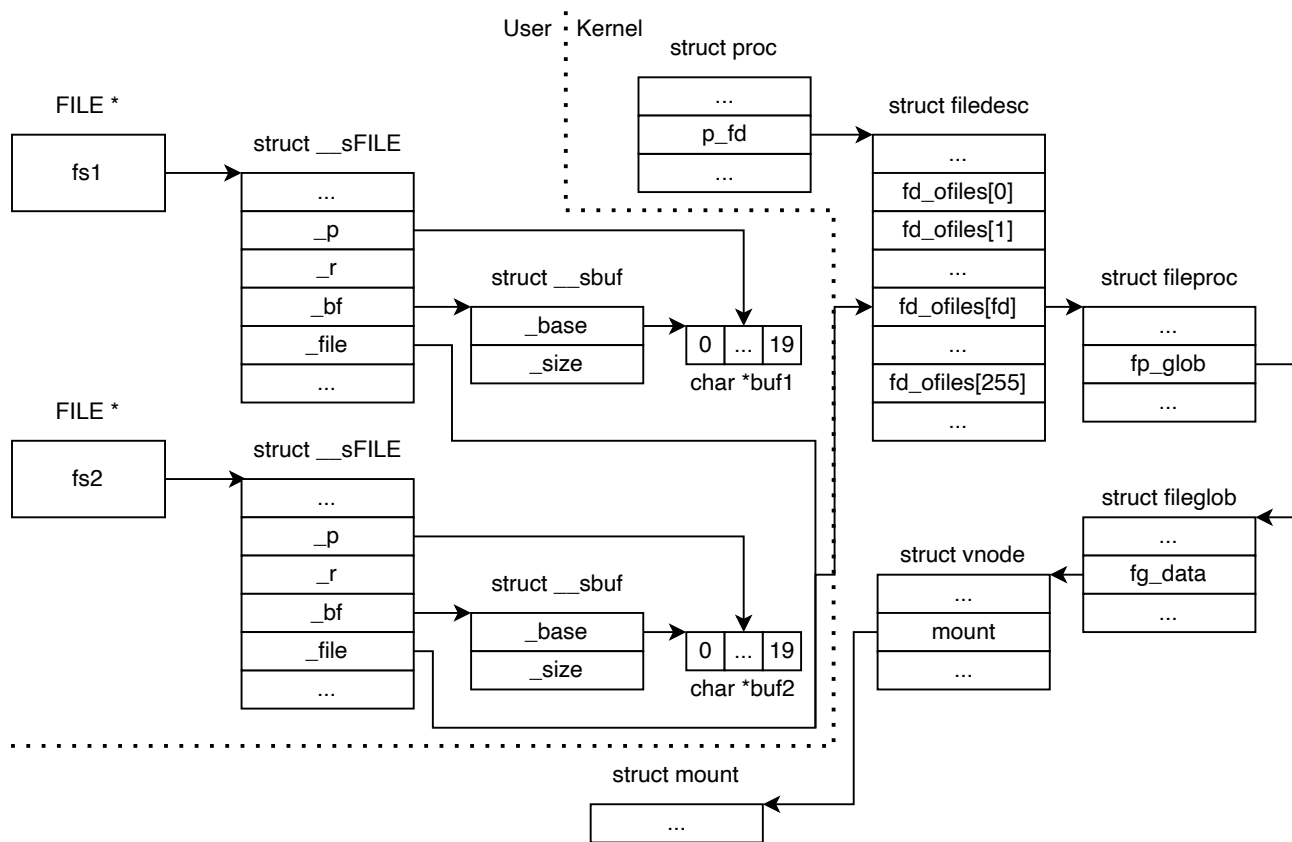


Рисунок 2 – Связь структур файловой подсистемы (BSD, Darwin-XNU)

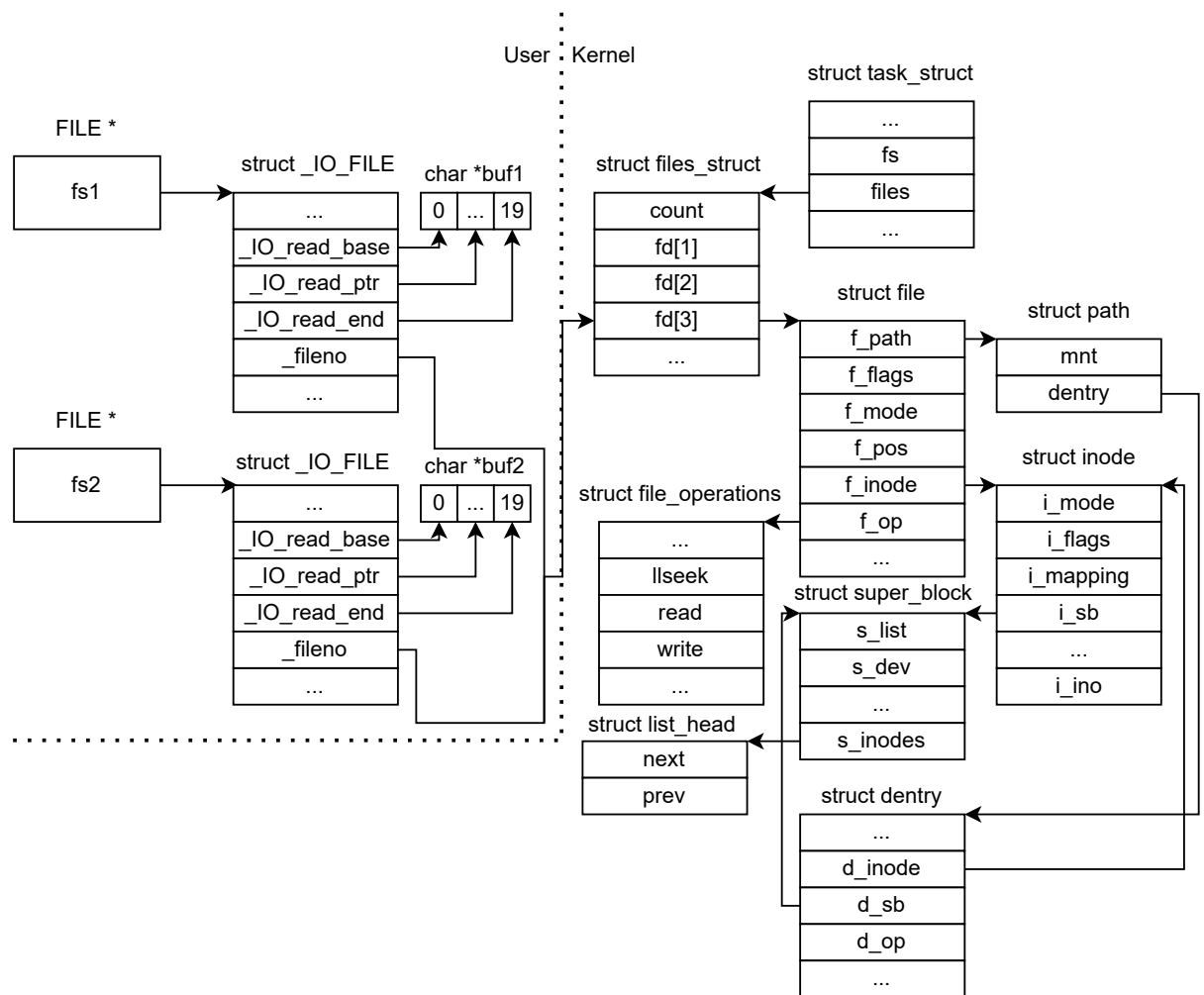


Рисунок 3 – Связь структур файловой подсистемы Linux

## Многопоточная

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <fcntl.h>
4 #include <pthread.h>
5 #include <unistd.h>
6
7 #define FILE_NAME    "../alphabet.txt"
8 #define BUFF_SIZE    20
9
10 #define RED          "\x1b[31m"
11 #define GREEN        "\x1b[32m"
12 #define RESET        "\x1b[0m"
13

```



```

14 void *fileReader(void *args) {
15     FILE *fs = (FILE *)args;
16     int rc = 1;
17     char c;
18     while (rc == 1) {
19         if ((rc = fscanf(fs, "%c\n", &c)) == 1)
20             fprintf(stdout, GREEN "%c" RESET, c);
21     }
22     return NULL;
23 }
24
25 int main(void) {
26     int fd = open(FILE_NAME, O_RDONLY);
27     if (fd == -1) {
28         perror("open");
29         exit(EXIT_FAILURE);
30     }
31     FILE *fs1 = fdopen(fd, "r");
32     char buf1[BUFF_SIZE];
33     setvbuf(fs1, buf1, _IOFBF, BUFF_SIZE);
34     FILE *fs2 = fdopen(fd, "r");
35     char buf2[BUFF_SIZE];
36     setvbuf(fs2, buf2, _IOFBF, BUFF_SIZE);
37     pthread_t thread;
38     int prc;
39     if ((prc = pthread_create(&thread, NULL, fileReader, (void
40         *)fs2)) != 0) {
41         fclose(fs1); fclose(fs2);
42         fprintf(stderr, "pthread_create_failed: %s\n",
43             strerror(prc));
44         exit(EXIT_FAILURE);
45     }
46     int rc = 1;
47     char c;
48     while (rc == 1) {
49         rc = fscanf(fs1, "%c\n", &c);
50         if (rc == 1)
51             fprintf(stdout, RED "%c" RESET, c);
52     }
53     if (pthread_join(thread, NULL) != 0) {
54         fclose(fs1); fclose(fs2);

```

```

53     fprintf(stderr, "pthread_create_failed: %s\n",
        strerror(prc));
54     exit(EXIT_FAILURE);
55 }
56 fclose(fs1);
57 fclose(fs2);
58 exit(EXIT_SUCCESS);
59 }

```

Листинг 3 – Первая программа (многопоточная)

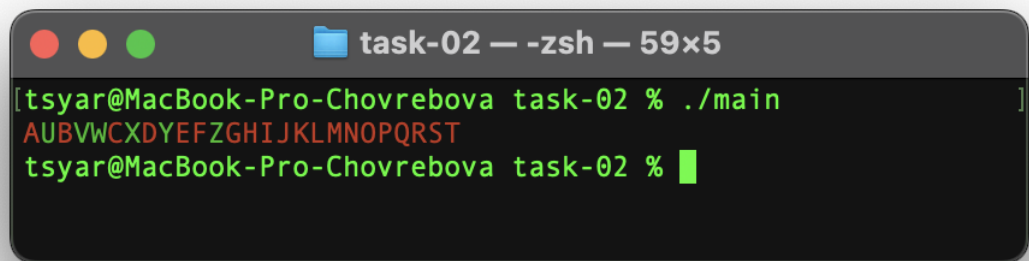


Рисунок 4 – Результат работы первой многопоточной программы

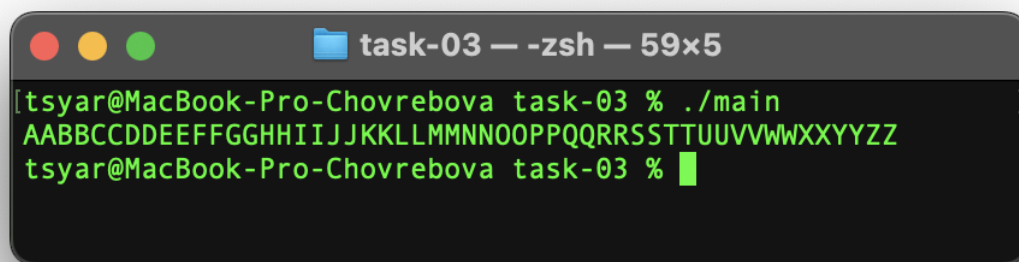
В многопоточной реализации программы вывод зависит от планирования потоков в системе. Первый поток чаще всего успевает отработать первым, из-за временных затрат на создание второго потока.

### 3 Программа №2

#### Однопоточная реализация

```
1 #include <stdlib.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4
5 #define FILE_NAME "../alphabet.txt"
6 #define CHAR_SIZE sizeof(char)
7 #define BUFF_SIZE 32 * CHAR_SIZE
8
9 int main() {
10     int fd1 = open(FILE_NAME, O_RDONLY);
11     if (fd1 == -1) {
12         write(STDERR_FILENO, "error:_open_fd1\n", BUFF_SIZE);
13         exit(EXIT_FAILURE);
14     }
15     int fd2 = open(FILE_NAME, O_RDONLY);
16     if (fd2 == -1) {
17         write(STDERR_FILENO, "error:_open_fd2\n", BUFF_SIZE);
18         close(fd1);
19         exit(EXIT_FAILURE);
20     }
21     int rc1 = 1, rc2 = 1;
22     char c;
23     while (rc1 == 1 || rc2 == 1) {
24         rc1 = read(fd1, &c, CHAR_SIZE);
25         if (rc1 == 1)
26             write(STDOUT_FILENO, &c, CHAR_SIZE);
27         rc2 = read(fd2, &c, 1);
28         if (rc2 == 1)
29             write(STDOUT_FILENO, &c, CHAR_SIZE);
30     }
31     write(STDOUT_FILENO, "\n", 1);
32     close(fd1);
33     close(fd2);
34     exit(EXIT_SUCCESS);
35 }
```

Листинг 4 – Вторая программа (однопоточная)

A terminal window titled "task-03 — -zsh — 59x5" on a dark background. The prompt is "tsyar@MacBook-Pro-Chovrebova task-03 %". The command ". /main" has been executed, resulting in the output "AABBCCDDEEFFGGHHIIJJKLLMMNN00PPQRRSSTTUUVVWXXYYZZ". The prompt is now "tsyar@MacBook-Pro-Chovrebova task-03 %" with a green cursor.

```
task-03 — -zsh — 59x5
[tsyar@MacBook-Pro-Chovrebova task-03 % ./main
AABBCCDDEEFFGGHHIIJJKLLMMNN00PPQRRSSTTUUVVWXXYYZZ
tsyar@MacBook-Pro-Chovrebova task-03 % ]
```

Рисунок 5 – Результат работы программы

### Анализ работы программы

Во время выполнения программы дважды вызывается системный вызов **open()**, который создает новый файловый дескриптор для открытого в режиме «только для чтения» файла «alphabet.txt» и возвращает созданный файловый дескриптор из системной таблицы открытых файлов процесса. Таким образом, создаются два независимых файловых дескриптора, указывающих на один и тот же файл. Для каждого дескриптора ядро создаёт отдельную структуру **struct file**, которая, помимо прочего, содержит поле **f\_pos** — текущее смещение в файле. Поскольку дескрипторы независимы, их смещения (**f\_pos**) также управляются независимо.

Далее в цикле поочередно считываются символы из файла и выводятся в стандартный поток вывода с использованием системного вызова **write()**. В результате каждый символ файла выводится дважды — поскольку оба дескриптора читают файл с начала, независимо друг от друга.

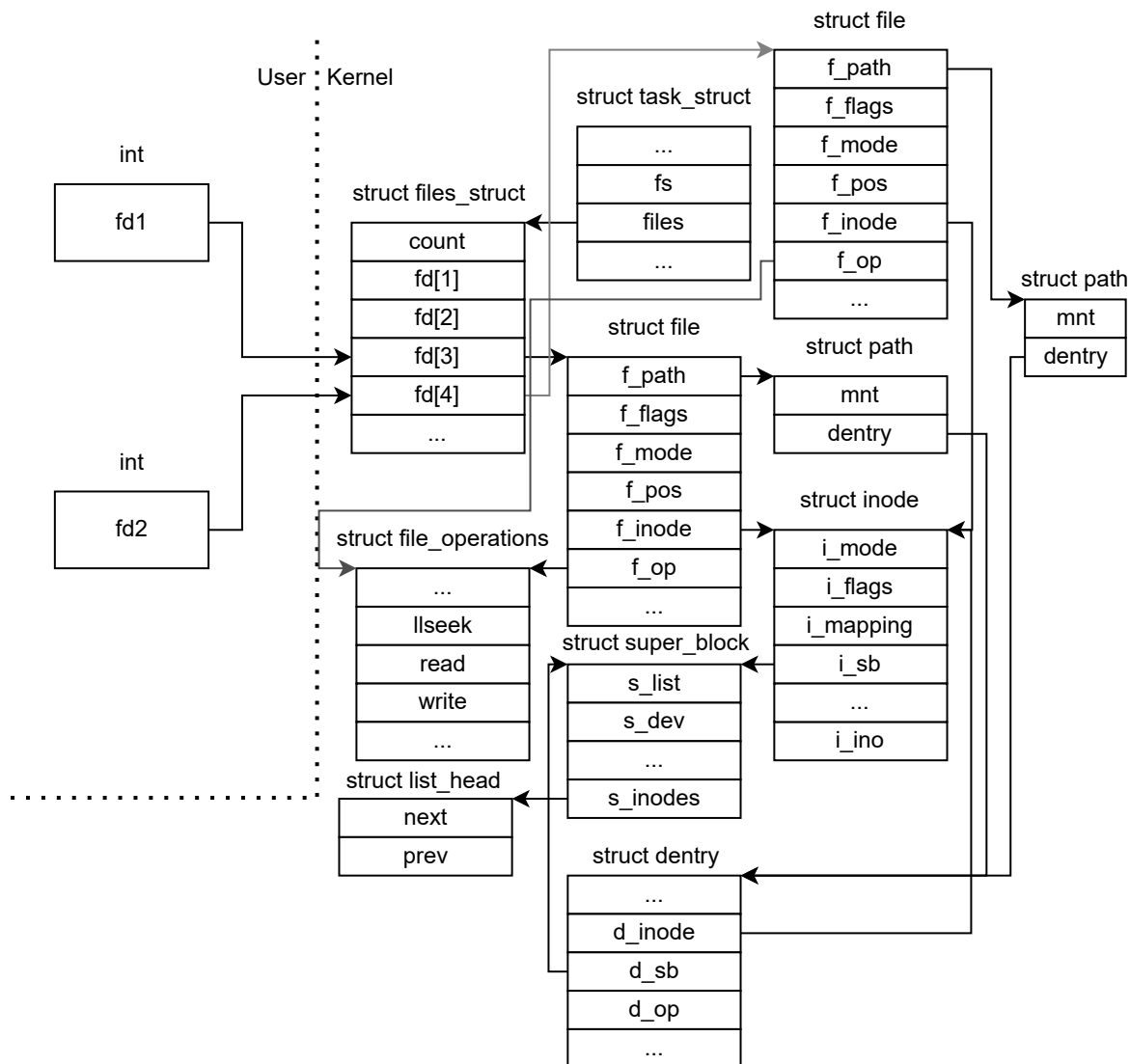


Рисунок 6 – Связь структур файловой подсистемы Linux

## Многопоточная реализация

```

1 #include <fcntl.h>
2 #include <unistd.h>
3 #include <pthread.h>
4
5 #define FILE_NAME "../alphabet.txt"
6 #define CHAR_SIZE sizeof(char)
7 #define BUFF_SIZE 32 * CHAR_SIZE
8
9 void *fileReader(void *args) {
10     int fd = *(int *)args;

```

```

11     int rc = 1;
12     char c;
13     while (rc == 1) {
14         rc = read(fd, &c, CHAR_SIZE);
15         if (rc == 1)
16             write(STDOUT_FILENO, &c, CHAR_SIZE);
17     }
18     return NULL;
19 }
20
21 int main() {
22     int fd1 = open(FILE_NAME, O_RDONLY);
23     if (fd1 == -1) {
24         write(STDERR_FILENO, "error:_open_fd1\n", BUFF_SIZE);
25         exit(EXIT_FAILURE);
26     }
27     int fd2 = open(FILE_NAME, O_RDONLY);
28     if (fd2 == -1) {
29         write(STDERR_FILENO, "error:_open_fd2\n", BUFF_SIZE);
30         close(fd1);
31         exit(EXIT_FAILURE);
32     }
33     pthread_t thread1, thread2;
34     int prc1, prc2;
35     if ((prc1 = pthread_create(&thread1, NULL, fileReader, &fd1))
        != 0) {
36         close(fd1); close(fd2);
37         write(STDERR_FILENO, "error:_pthread1_create\n",
            BUFF_SIZE);
38         exit(EXIT_FAILURE);
39     }
40     if ((prc2 = pthread_create(&thread2, NULL, fileReader, &fd2))
        != 0) {
41         close(fd1); close(fd2);
42         write(STDERR_FILENO, "error:_pthread2_create\n",
            BUFF_SIZE);
43         exit(EXIT_FAILURE);
44     }
45     if (pthread_join(thread1, NULL)) {
46         close(fd1); close(fd2);
47         write(STDERR_FILENO, "error:_pthread1_join\n", BUFF_SIZE);

```

```

48         exit(EXIT_FAILURE);
49     }
50     if (pthread_join(thread2, NULL)) {
51         close(fd1); close(fd2);
52         write(STDERR_FILENO, "error:_pthread2_join\n", BUFF_SIZE);
53         exit(EXIT_FAILURE);
54     }
55     close(fd1);
56     close(fd2);
57     exit(EXIT_SUCCESS);
58 }

```

Листинг 5 – Многопоточная программа с двумя дополнительными потоками

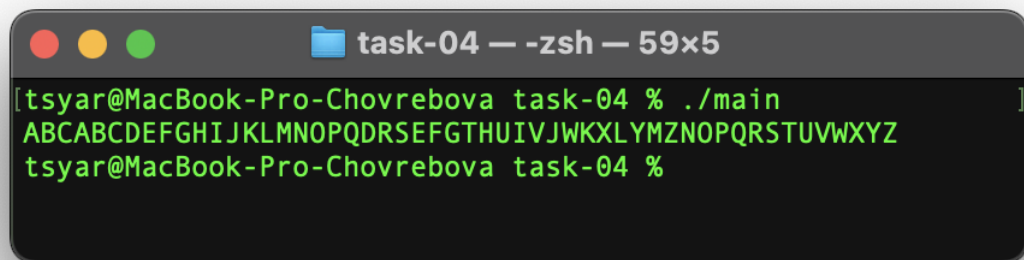


Рисунок 7 – Результат работы второй многопоточной программы

В многопоточной реализации системный вызов **open()** также дважды используется для открытия одного и того же файла «alphabet.txt» в режиме **O\_RDONLY**. В результате создаются два независимых файловых дескриптора, каждый из которых связан с собственной структурой **struct file** в ядре. Эти структуры содержат отдельные значения текущего смещения в файле, что обеспечивает независимое перемещение указателя **f\_pos** для каждого дескриптора.

Однако, в отличие от однопоточной версии, где чтение из обоих дескрипторов осуществляется последовательно в рамках одного потока, многопоточная реализация организует параллельную обработку с использованием двух дополнительных потоков, создаваемых с помощью **pthread\_create()**. Каждый поток получает один из файловых дескрипторов в качестве аргу-

мента и выполняет функцию **fileReader()**, которая реализует цикл чтения по одному символу из файла и немедленного вывода его в стандартный поток вывода (**STDOUT\_FILENO**) с помощью системного вызова **write()**.

```
1 #include <fcntl.h>
2 #include <unistd.h>
3 #include <pthread.h>
4
5 #define FILE_NAME "../alphabet.txt"
6 #define CHAR_SIZE sizeof(char)
7 #define BUFF_SIZE 32 * CHAR_SIZE
8
9 void *fileReader(void *args) {
10     int fd = *(int *)args;
11     int rc = 1; char c;
12     while (rc == 1) {
13         rc = read(fd, &c, CHAR_SIZE);
14         if (rc == 1)
15             write(STDOUT_FILENO, &c, CHAR_SIZE);
16     }
17     return NULL;
18 }
19
20 int main() {
21     int fd1 = open(FILE_NAME, O_RDONLY);
22     if (fd1 == -1) {
23         write(STDERR_FILENO, "error:_open_fd1\n", BUFF_SIZE);
24         exit(EXIT_FAILURE);
25     }
26     int fd2 = open(FILE_NAME, O_RDONLY);
27     if (fd2 == -1) {
28         write(STDERR_FILENO, "error:_open_fd2\n", BUFF_SIZE);
29         close(fd1);
30         exit(EXIT_FAILURE);
31     }
32     pthread_t thread; int prc;
33     if ((prc = pthread_create(&thread, NULL, fileReader, &fd1))
34         != 0) {
35         close(fd1); close(fd2);
36         write(STDERR_FILENO, "error:_pthread1_create\n",
37             BUFF_SIZE);
38         exit(EXIT_FAILURE);
39     }
```



```

37     }
38     fileReader(&fd2);
39     if (pthread_join(thread, NULL)) {
40         close(fd1); close(fd2);
41         write(STDERR_FILENO, "error:_pthread_join\n", BUFF_SIZE);
42         exit(EXIT_FAILURE);
43     }
44     close(fd1); close(fd2);
45     exit(EXIT_SUCCESS);
46 }

```

Листинг 6 – Многопоточная программа с одним дополнительным потоком

В случае многопоточной программы, алфавит будет выведен 2 раза, но порядок вывода символов заранее предсказать невозможно.

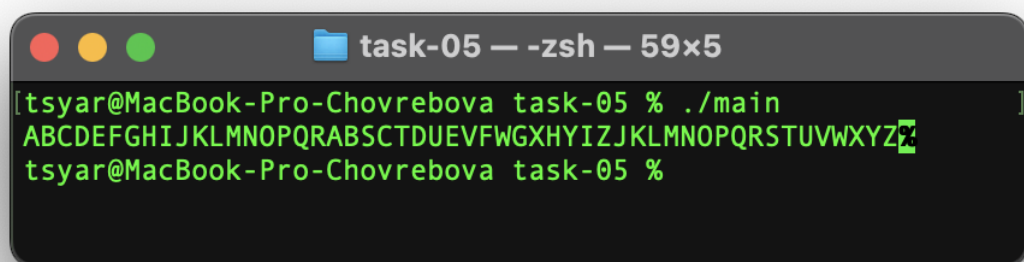


Рисунок 8 – Результат работы второй многопоточной программы

В данной реализации также создаются два файловых дескриптора, но один обрабатывается в основном потоке, а второй – в отдельном созданном потоке. Как и ранее, вывод символов может перемешиваться из-за отсутствия синхронизации между потоками.

## 4 Программа №3

### Однопоточная реализация

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <fcntl.h>
4 #include <unistd.h>
5 #include <sys/stat.h>
6 #define FILE_OUT "output.txt"
7
8 void fileInfo(FILE *f) {
9     struct stat statbuf;
10    stat(FILE_OUT, &statbuf);
11    printf("st_ino:_%llu_|_", statbuf.st_ino);
12    printf("st_size:_%d_|_", statbuf.st_blksize);
13    printf("pos:_%ld\n", ftell(f));
14 }
15
16 int main() {
17     FILE *f1 = fopen(FILE_OUT, "w");
18     fileInfo(f1);
19     FILE *f2 = fopen(FILE_OUT, "w");
20     fileInfo(f2);
21     for (char c = 'A'; c <= 'Z'; c++) {
22         if (c % 2) {
23             fprintf(f1, "%c", c);
24             fileInfo(f1);
25         } else {
26             fprintf(f2, "%c", c);
27             fileInfo(f2);
28         }
29     }
30     fclose(f1);
31     fileInfo(f1);
32     fclose(f2);
33     fileInfo(f2);
34     exit(EXIT_SUCCESS);
35 }
```

Листинг 7 – Однопоточная программа

```
task-06 — -zsh — 59x34
[tsyar@MacBook-Pro-Chovrebova task-06 % ./main ]
st_ino: 28745001 | st_size: 4096 | pos: 0
st_ino: 28745001 | st_size: 4096 | pos: 0
st_ino: 28745001 | st_size: 4096 | pos: 1
st_ino: 28745001 | st_size: 4096 | pos: 1
st_ino: 28745001 | st_size: 4096 | pos: 2
st_ino: 28745001 | st_size: 4096 | pos: 2
st_ino: 28745001 | st_size: 4096 | pos: 3
st_ino: 28745001 | st_size: 4096 | pos: 3
st_ino: 28745001 | st_size: 4096 | pos: 4
st_ino: 28745001 | st_size: 4096 | pos: 4
st_ino: 28745001 | st_size: 4096 | pos: 5
st_ino: 28745001 | st_size: 4096 | pos: 5
st_ino: 28745001 | st_size: 4096 | pos: 6
st_ino: 28745001 | st_size: 4096 | pos: 6
st_ino: 28745001 | st_size: 4096 | pos: 7
st_ino: 28745001 | st_size: 4096 | pos: 7
st_ino: 28745001 | st_size: 4096 | pos: 8
st_ino: 28745001 | st_size: 4096 | pos: 8
st_ino: 28745001 | st_size: 4096 | pos: 9
st_ino: 28745001 | st_size: 4096 | pos: 9
st_ino: 28745001 | st_size: 4096 | pos: 10
st_ino: 28745001 | st_size: 4096 | pos: 10
st_ino: 28745001 | st_size: 4096 | pos: 11
st_ino: 28745001 | st_size: 4096 | pos: 11
st_ino: 28745001 | st_size: 4096 | pos: 12
st_ino: 28745001 | st_size: 4096 | pos: 12
st_ino: 28745001 | st_size: 4096 | pos: 13
st_ino: 28745001 | st_size: 4096 | pos: 13
st_ino: 28745001 | st_size: 4096 | pos: -1
st_ino: 28745001 | st_size: 4096 | pos: -1
[tsyar@MacBook-Pro-Chovrebova task-06 % cat output.txt ]
BDFHJLNPRTVXZ
[tsyar@MacBook-Pro-Chovrebova task-06 % ]
```

Рисунок 9 – Результат работы третьей однопоточной программы

### Анализ работы программы

Во время выполнения программы дважды вызывается библиотечная функция **fopen()** с режимом "w", что соответствует открытию файла «output.txt» только на запись с предварительным обнулением содержимого. Каждый вызов **fopen()** создает новый файловый дескриптор и инициализирует связан-

ную с ним структуру `_IO_FILE`, управляющую буферизованным вводом-выводом на уровне стандартной библиотеки языка C, и возвращает указатель типа **FILE** на эту структуру.

Фактическая запись в файл происходит только при вызове **fflush()** или **fclose()**, в соответствии с механизмом буферизации стандартной библиотеки.

Во время выполнения программы символы английского алфавита поочередно записываются в разные буферы: нечётные символы направляются в поток `f1`, чётные – в поток `f2`. Однако до момента вызова **fclose()** содержимое буферов не попадает в файл. После закрытия потока `f1` его буфер сбрасывается в файл, и в него записываются нечётные символы. Затем, при закрытии потока `f2`, его буфер записывается начиная с начала файла (так как файловый дескриптор у потока `f2` имел своё смещение), в результате чего данные, записанные через `f1`, оказываются перезаписанными чётными символами.

## Многопоточная

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <fcntl.h>
4 #include <pthread.h>
5 #include <unistd.h>
6 #include <sys/stat.h>
7
8 #define FILE_OUT "output.txt"
9
10 struct arg {
11     FILE *f;
12     int off;
13 };
14 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
15
16 void fileInfo(FILE *f) {
17     struct stat statbuf;
18     stat(FILE_OUT, &statbuf);
19     pthread_mutex_lock(&mutex);
20     printf("st_ino:_%llu_|_", statbuf.st_ino);
21     printf("st_size:_%d_|_", statbuf.st_blksize);
22     printf("pos:_%ld\n", ftell(f));
23     pthread_mutex_unlock(&mutex);
```

```

24 }
25
26 void *fileWriter(void *arg) {
27     struct arg *args = (struct arg*) arg;
28     for (char c = 'A' + args->off; c <= 'Z'; c+=2) {
29         fprintf(args->f, "%c", c);
30         fileInfo(args->f);
31     }
32     return NULL;
33 }
34
35 int main() {
36     FILE *f1 = fopen(FILE_OUT, "w");
37     fileInfo(f1);
38     FILE *f2 = fopen(FILE_OUT, "w");
39     fileInfo(f2);
40     pthread_t thread1, thread2;
41     struct arg arg1 = {.f = f1, .off = 0};
42     struct arg arg2 = {.f = f2, .off = 1};
43     if (pthread_create(&thread1, NULL, fileWriter, &arg1) != 0) {
44         perror("error:_create_thread1");
45         fclose(f1); fclose(f2);
46         exit(EXIT_FAILURE);
47     }
48     if (pthread_create(&thread2, NULL, fileWriter, &arg2) != 0) {
49         perror("error:_create_thread2");
50         fclose(f1); fclose(f2);
51         exit(EXIT_FAILURE);
52     }
53     pthread_join(thread1, NULL);
54     pthread_join(thread2, NULL);
55     fclose(f1);
56     fileInfo(f1);
57     fclose(f2);
58     fileInfo(f2);
59     exit(EXIT_SUCCESS);
60 }

```

Листинг 8 – Многопоточная программа с двумя дополнительными потоками

Отличие в работе многопоточной программы от однопоточной в том, что данные записываются в буферы в непредсказуемом порядке.

```
task-07 — -zsh — 59x34
[tsyar@MacBook-Pro-Chovrebova task-07 % ./main
st_ino: 28745966 | st_size: 4096 | pos: 0
st_ino: 28745966 | st_size: 4096 | pos: 0
st_ino: 28745966 | st_size: 4096 | pos: 1
st_ino: 28745966 | st_size: 4096 | pos: 2
st_ino: 28745966 | st_size: 4096 | pos: 3
st_ino: 28745966 | st_size: 4096 | pos: 4
st_ino: 28745966 | st_size: 4096 | pos: 5
st_ino: 28745966 | st_size: 4096 | pos: 6
st_ino: 28745966 | st_size: 4096 | pos: 7
st_ino: 28745966 | st_size: 4096 | pos: 8
st_ino: 28745966 | st_size: 4096 | pos: 9
st_ino: 28745966 | st_size: 4096 | pos: 1
st_ino: 28745966 | st_size: 4096 | pos: 2
st_ino: 28745966 | st_size: 4096 | pos: 3
st_ino: 28745966 | st_size: 4096 | pos: 4
st_ino: 28745966 | st_size: 4096 | pos: 10
st_ino: 28745966 | st_size: 4096 | pos: 5
st_ino: 28745966 | st_size: 4096 | pos: 6
st_ino: 28745966 | st_size: 4096 | pos: 7
st_ino: 28745966 | st_size: 4096 | pos: 8
st_ino: 28745966 | st_size: 4096 | pos: 11
st_ino: 28745966 | st_size: 4096 | pos: 12
st_ino: 28745966 | st_size: 4096 | pos: 13
st_ino: 28745966 | st_size: 4096 | pos: 9
st_ino: 28745966 | st_size: 4096 | pos: 10
st_ino: 28745966 | st_size: 4096 | pos: 11
st_ino: 28745966 | st_size: 4096 | pos: 12
st_ino: 28745966 | st_size: 4096 | pos: 13
st_ino: 28745966 | st_size: 4096 | pos: -1
st_ino: 28745966 | st_size: 4096 | pos: -1
[tsyar@MacBook-Pro-Chovrebova task-07 % cat output.txt
BDFHJLNPRTVXZ
tsyar@MacBook-Pro-Chovrebova task-07 %
```

Рисунок 10 – Результат работы третьей многопоточной программы

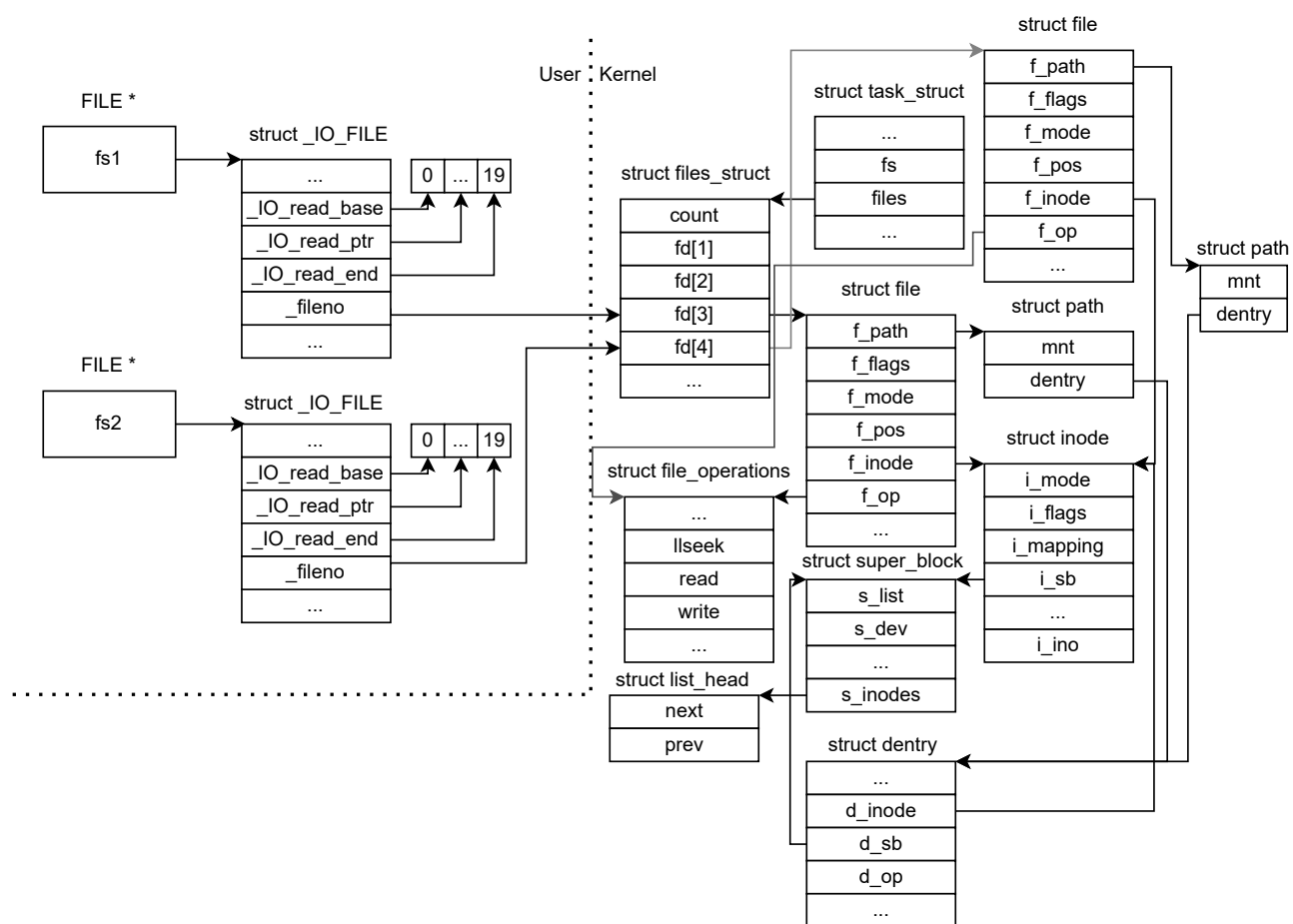


Рисунок 11 – Связь структур файловой подсистемы Linux