



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»

Отчёт по лабораторной работе  
по дисциплине «Операционные системы»

Тема: Буферизованный и небуферизованный ввод-вывод

Студент: Княжев А. В.

Группа: ИУ7-62Б

Оценка (баллы): \_\_\_\_\_

Преподаватель: Рязанова Н. Ю.

Москва — 2023 г.

# 1. Структуры

## 1.1. `_IO_FILE`

```
1 typedef struct _IO_FILE FILE;
2 struct _IO_FILE
3 {
4     int _flags;    /* High-order word is _IO_MAGIC; rest is flags. */
5
6     /* The following pointers correspond to the C++ streambuf protocol. */
7     char *_IO_read_ptr; /* Current read pointer */
8     char *_IO_read_end; /* End of get area. */
9     char *_IO_read_base; /* Start of putback+get area. */
10    char *_IO_write_base; /* Start of put area. */
11    char *_IO_write_ptr; /* Current put pointer. */
12    char *_IO_write_end; /* End of put area. */
13    char *_IO_buf_base; /* Start of reserve area. */
14    char *_IO_buf_end; /* End of reserve area. */
15
16    /* The following fields are used to support backing up and undo. */
17    char *_IO_save_base; /* Pointer to start of non-current get area. */
18    char *_IO_backup_base; /* Pointer to first valid character of backup
19                             area */
20
21    char *_IO_save_end; /* Pointer to end of non-current get area. */
22
23    struct _IO_marker *_markers;
24
25    struct _IO_FILE *_chain;
26
27    int _fileno;
28    int _flags2;
29    __off_t _old_offset; /* This used to be _offset but it's too small. */
30
31    /* 1+column number of pbase(); 0 is unknown. */
32    unsigned short _cur_column;
33    signed char _vtable_offset;
```

```

32  char _shortbuf[1];
33
34  _IO_lock_t *_lock;
35  #ifdef _IO_USE_OLD_IO_FILE
36  };

```

## 1.2. 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 реализация

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #define BUF_SIZE 20
4 #define FILENAME "alphabet.txt"
5 int main()
6 {
7     int fd = open(FILENAME, O_RDONLY);
8     FILE* fs1 = fdopen(fd, "r");
9     char buff1[BUF_SIZE];
10    setvbuf(fs1, buff1, _IOFBF, BUF_SIZE);
11    FILE* fs2 = fdopen(fd, "r");
12    char buff2[BUF_SIZE];
13    setvbuf(fs2, buff2, _IOFBF, BUF_SIZE);
14    int flag1 = 1, flag2 = 2;
15    while(flag1 == 1 || flag2 == 1)
16    {
17        char c;
18        flag1 = fscanf(fs1, "%c", &c);
19        if (flag1 == 1)
20        {
21            fprintf(stdout, "%c", c);
22        }
23        flag2 = fscanf(fs2, "%c", &c);
24        if (flag2 == 1)
25        {
26            fprintf(stdout, "%c", c);
27        }
28    }
29    return 0;
30 }
```

Вывод программы

```
muhomorfus@mikrovolnowka:~/Bamonka/6/os-labs/lab_08

♥ muhomorfus: ~/B/6/o/Lab_08 → ./1_1 11:55:17
aubvcwdxeyfzghijklmnopqrst%
♥ muhomorfus: ~/B/6/o/Lab_08 → ./1_1 11:55:21
aubvcwdxeyfzghijklmnopqrst%
♥ muhomorfus: ~/B/6/o/Lab_08 → | 11:55:27
```

## 2.2. Многопоточная реализация

```
1 #include <fcntl.h>
2 #include <pthread.h>
3 #include <stdio.h>
4 #define BUF_SIZE 20
5 #define FILENAME "alphabet.txt"
6 void* thread1(void *args)
7 {
8     int* fd = (int*)args;
9     FILE* fs1 = fdopen(*fd, "r");
10    char buff1[BUF_SIZE];
11    setvbuf(fs1, buff1, _IOFBF, BUF_SIZE);
12    int flag = 1;
13    char c;
14    while ((flag = fscanf(fs1, "%c", &c)) == 1)
15    {
16        fprintf(stdout, "%c", c);
17    }
18    return NULL;
19 }
20 void* thread2(void* args)
21 {
22    int* fd = (int*)args;
23    FILE* fs2 = fdopen(*fd, "r");
24    char buff2[BUF_SIZE];
```

```

25     setvbuf ( fs2 , buff2 , _IOFBF, BUF_SIZE);
26     int flag = 1;
27     char c;
28     while (( flag = fscanf(fs2 , "%c" , &c)) == 1)
29     {
30         fprintf(stdout , "%c" , c);
31     }
32     return NULL;
33 }
34 int main()
35 {
36     pthread_t t1 , t2 ;
37     int fd = open(FILENAME, O_RDONLY);
38     pthread_create(&t1 , NULL, thread1 , &fd);
39     pthread_create(&t2 , NULL, thread2 , &fd);
40     pthread_join(t1 , NULL);
41     pthread_join(t2 , NULL);
42     return 0 ;
43 }

```

## Вывод программы

```

muhomorfus@mikrovolnowka:~/Bamonka/6/os-labs/lab_08
♥ muhomorfus: ~/B/6/o/lab_08 → ./1_2 11:56:03
abcdefghijklmnopqrstuvwxyz
♥ muhomorfus: ~/B/6/o/lab_08 → ./1_2 11:56:05
abcdefghijklmnopqrstuvwxyz
♥ muhomorfus: ~/B/6/o/lab_08 → ./1_2 11:56:06
abcdefghijklmnopqrstuvwxyz
♥ muhomorfus: ~/B/6/o/lab_08 → | 11:56:07

```

## 2.3. Выводы

В программе файл открывается 1 раз системным вызовом `open()`, который возвращает номер дескриптора. Возвращенный номер — индекс дескриптора открытого файла в

таблице дескрипторов открытых файлов процесса.

Затем 2 раза вызывается функция `fdopen()`, которая создаёт указатель на структуру `FILE`, определённую с помощью `define` на базе `struct _IO_FILE`.

Функции `fdopen()` нужно передать возвращённый из `open()` дескриптор `fd`. Поле `_fileno` в `struct _IO_FILE` содержит номер этого дескриптора.

Функция `setvbuf()` устанавливает размер буфера 20 байт.

При первом вызове функции `fscanf()` в цикле (для `fs1`) `buff1` будет заполнен полностью — первыми 20 символами (буквами латинского алфавита). `f_pos` в структуре `struct_file` открытого файла увеличится на 20.

При втором вызове `fscanf()` в цикле (для `fs2`) буфер `buff2` будет заполнен оставшимися 6 символами (начиная с `f_pos`, изменённого после 1-ого вызова `fscanf()`).

В случае многопоточной реализации потоки выполняются с разной скоростью, поэтому символы перемешаются.

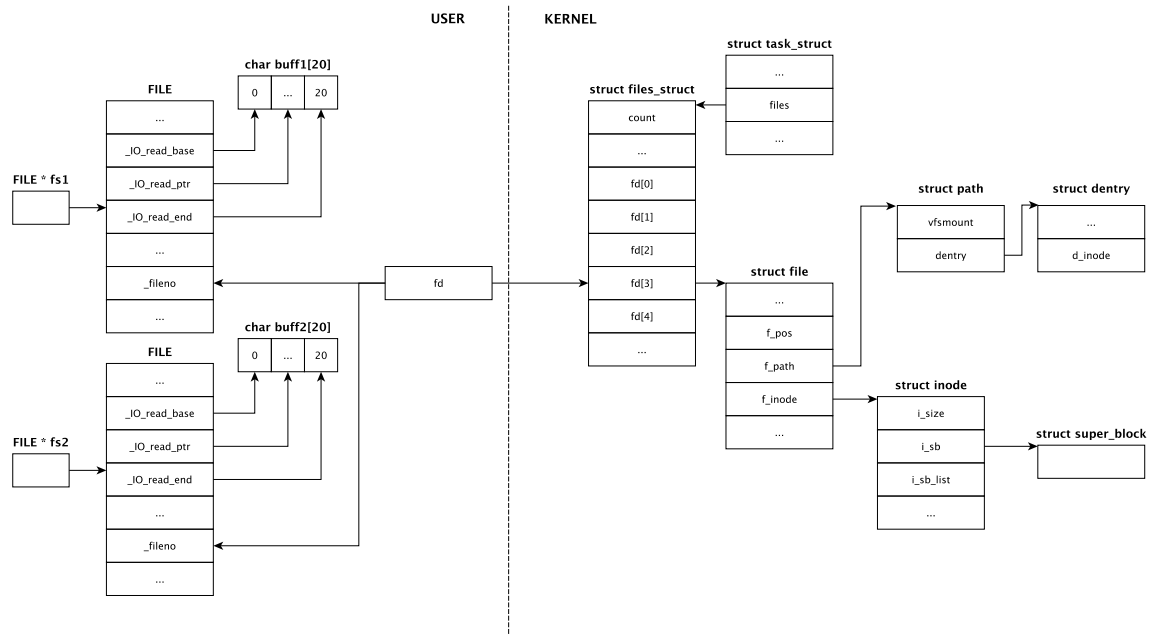


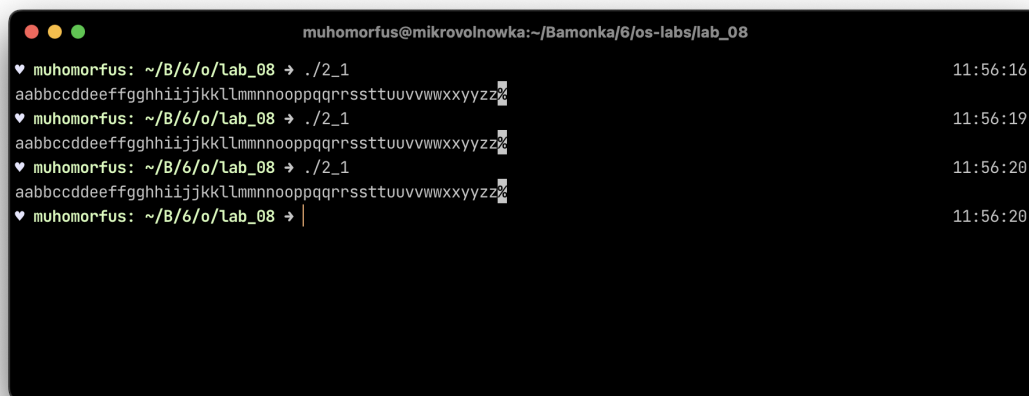
Рисунок 2.1 — Используемые структуры

## 3. Программа 2

### 3.1. Однопоточная реализация

```
1 #include <fcntl.h>
2 #include <unistd.h>
3 int main()
4 {
5     char c;
6     int fd1 = open("alphabet.txt", O_RDONLY);
7     int fd2 = open("alphabet.txt", O_RDONLY);
8     while((read(fd1, &c, 1) == 1) && (read(fd2, &c, 1) == 1))
9     {
10         write(1, &c, 1);
11         write(1, &c, 1);
12     }
13     return 0;
14 }
```

Вывод программы



```
muhomorfus@mikrovolnowka:~/Bamonka/6/os-labs/lab_08
♥ muhomorfus: ~/B/6/o/Lab_08 → ./2_1 11:56:16
aabccddeeffgghhiijjkkllmmnnoppqrrssttuuvvwxxyyzz
♥ muhomorfus: ~/B/6/o/Lab_08 → ./2_1 11:56:19
aabccddeeffgghhiijjkkllmmnnoppqrrssttuuvvwxxyyzz
♥ muhomorfus: ~/B/6/o/Lab_08 → ./2_1 11:56:20
aabccddeeffgghhiijjkkllmmnnoppqrrssttuuvvwxxyyzz
♥ muhomorfus: ~/B/6/o/Lab_08 → 11:56:20
```

### 3.2. Многопоточная реализация



```

1 #include <fcntl.h>
2 #include <pthread.h>
3 #include <unistd.h>
4 #include <stdio.h>
5 #define BUF_SIZE 20
6 #define FILENAME "alphabet.txt"
7 void* thread1()
8 {
9     int fd = open(FILENAME, O_RDONLY);
10
11     char c;
12     while ((read(fd, &c, 1)) == 1) write(1, &c, 1);
13
14     return NULL;
15 }
16 void* thread2()
17 {
18     FILE* f = fopen(FILENAME, "w");
19
20     for (char c = 'b'; c <= 'z'; c += 2)
21     {
22         fprintf(f, "%c", c);
23     }
24     fclose(f);
25     return NULL;
26 }
27 int main()
28 {
29     pthread_t t1, t2;
30     pthread_create(&t1, NULL, thread1, NULL);
31     pthread_create(&t2, NULL, thread2, NULL);
32     pthread_join(t1, NULL);
33     pthread_join(t2, NULL);
34     return 0;
35 }

```

**Вывод программы**

```
muhomorfus@mikrovolnowka:~/Bamonka/6/os-labs/lab_08
♥ muhomorfus: ~/B/6/o/Lab_08 → ./2_2 11:56:25
aabcdbcedfegfghijhijklmnlmonpoqprqsntsutvuvwxxyz%
♥ muhomorfus: ~/B/6/o/Lab_08 → ./2_2 11:56:27
aabcdbcedfegfghiijjkkllmnlmonpoqprqsntsutvuvwxxyz%
♥ muhomorfus: ~/B/6/o/Lab_08 → ./2_2 11:56:28
aabcdbceddeeffgghiijjkkllmnlmonpoqprqssttuuvvwxxyz%
♥ muhomorfus: ~/B/6/o/Lab_08 → | 11:56:28
```

### 3.3. Выводы

Функция `open()` два раза создает дескриптор для одного и того же файла в системной таблице открытых файлов, поэтому в программе существует два различных дескриптора открытого файла (`struct file`), ссылающихся на один и тот же `struct inode`.

Так как у каждой структуры `struct file` свое поле `f_pos`, выводимые символы будут дублироваться.

В случае многопоточной реализации потоки выполняются с разной скоростью, поэтому символы перемешаются.

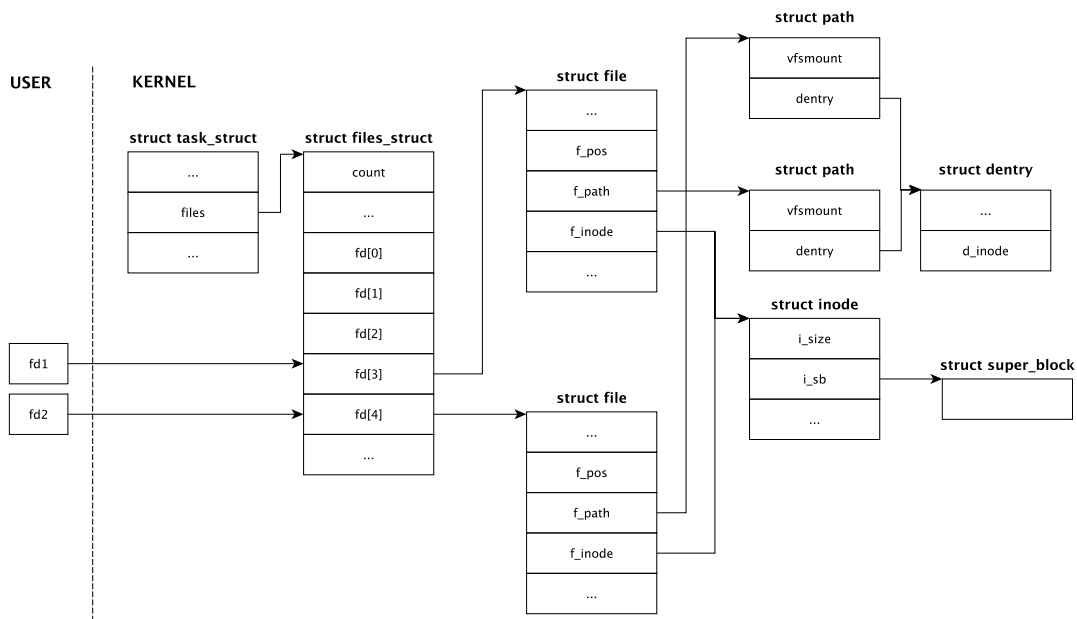


Рисунок 3.1 — Используемые структуры

## 4. Программа 3

### 4.1. fopen()

#### 4.1.1. Однопоточная реализация

```
1 #include <fcntl.h>
2 #include <stdio.h>
3 #include <sys/types.h>
4 #include <sys/stat.h>
5 #include <unistd.h>
6 #define FILENAME "tmp1.txt"
7 int main()
8 {
9     struct stat buf;
10    FILE* f1 = fopen(FILENAME, "w");
11    int rc = fstat(f1->_file, &buf);
12    if (!rc)
13    {
14        fprintf(stdout, "after_fopen_f1:_inode_-%ju,_total_size_-%lld\n", (
            uintmax_t)buf.st_ino, buf.st_size);
15    }
16    FILE* f2 = fopen(FILENAME, "w");
17    rc = fstat(f2->_file, &buf);
18    if (!rc)
19    {
20        fprintf(stdout, "after_fopen_f2:_inode_-%ju,_total_size_-%lld\n", (
            uintmax_t)buf.st_ino, buf.st_size);
21    }
22    for (char c = 'a'; c <= 'z'; c++)
23    {
24        if (c % 2)
25        {
26            fprintf(f1, "%c", c);
27        }
28        else
```

```

29     {
30         fprintf(f2, "%c", c);
31     }
32 }
33 rc = fstat(f1->_file, &buf);
34 if (!rc)
35 {
36     fprintf(stdout, "before_fclose_f1:_inode_-%ju,_total_size_-%lld\n",
37             (uintmax_t)buf.st_ino, buf.st_size);
38 }
39 rc = fstat(f2->_file, &buf);
40 if (!rc)
41 {
42     fprintf(stdout, "before_fclose_f2:_inode_-%ju,_total_size_-%lld\n",
43             (uintmax_t)buf.st_ino, buf.st_size);
44 }
45 fclose(f1);
46 rc = stat(FILENAME, &buf);
47 fprintf(stdout, "after_fclose_f1:_inode_-%ju,_total_size_-%lld\n", (
48     uintmax_t)buf.st_ino, buf.st_size);
49 fclose(f2);
50 rc = stat(FILENAME, &buf);
51 fprintf(stdout, "after_fclose_f2:_inode_-%ju,_total_size_-%lld\n", (
52     uintmax_t)buf.st_ino, buf.st_size);
53 return 0;
54 }

```

## Вывод программы

```
muhomorfus@mikrovolnowka:~/Bamonka/6/os-labs/lab_08

♥ muhomorfus: ~/B/6/o/Lab_08 → ./3_fopen_1 11:56:36
after fopen f1: inode - 26011859, total size - 0
after fopen f2: inode - 26011859, total size - 0
before fclose f1: inode - 26011859, total size - 0
before fclose f2: inode - 26011859, total size - 0
after fclose f1: inode - 26011859, total size - 13
after fclose f2: inode - 26011859, total size - 13
♥ muhomorfus: ~/B/6/o/Lab_08 → cat tmp1.txt 11:56:41
bdfhjlnprtvxz
♥ muhomorfus: ~/B/6/o/Lab_08 → | 11:57:07
```

#### 4.1.2. Многопоточная реализация

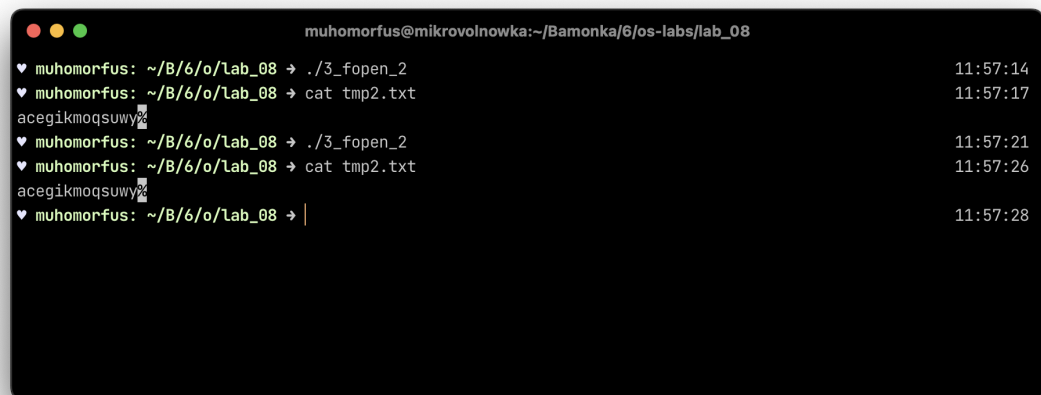
```
1  #include <fcntl.h>
2  #include <pthread.h>
3  #include <stdio.h>
4  #include <unistd.h>
5  #define FILENAME "tmp2.txt"
6  void* thread1()
7  {
8      FILE* f = fopen(FILENAME, "w");
9      for (char c = 'a'; c <= 'z'; c += 2)
10     {
11         fprintf(f, "%c", c);
12     }
13     fclose(f);
14     return NULL;
15 }
16 void* thread2()
17 {
18     FILE* f = fopen(FILENAME, "w");
19     for (char c = 'b'; c <= 'z'; c += 2)
20     {
21         fprintf(f, "%c", c);
22     }
23     fclose(f);
24     return NULL;
```

```

25 }
26 int main()
27 {
28     pthread_t t1, t2;
29     pthread_create(&t1, NULL, thread1, NULL);
30     pthread_create(&t2, NULL, thread2, NULL);
31     pthread_join(t1, NULL);
32     pthread_join(t2, NULL);
33     return 0;
34 }

```

### Вывод программы



```

muhomorfus@mikrovolnowka:~/Bamonka/6/os-labs/lab_08
♥ muhomorfus: ~/B/6/o/lab_08 → ./3_fopen_2 11:57:14
♥ muhomorfus: ~/B/6/o/lab_08 → cat tmp2.txt 11:57:17
acegikmoqsuwyz
♥ muhomorfus: ~/B/6/o/lab_08 → ./3_fopen_2 11:57:21
♥ muhomorfus: ~/B/6/o/lab_08 → cat tmp2.txt 11:57:26
acegikmoqsuwyz
♥ muhomorfus: ~/B/6/o/lab_08 → | 11:57:28

```

### 4.1.3. Выводы

Файл открывается на запись два раза функцией `fopen()`, которая внутри своего кода вызывает `open()`. Каждое открытие файла сопровождается созданием дескриптора открытого файла `struct file`, содержащего поле `f_pos`.

Поля `f_pos` двух разных дескрипторов одного открытого файла независимы, поэтому запись в файл будет производиться с нулевой позиции.

Функция `fprintf()` предоставляет буферизованный вывод, поэтому сначала информация пишется в буфер, а из буфера в файл при одном из условий:

1. буфер заполнен;
2. вызвана функция `fclose()`;
3. вызвана функция `fflush()` (принудительная запись).

При вызове `fclose()` для `fs1` буфер для `fs1` записывается в файл. При вызове `fclose()` для `fs2`, все содержимое файла затирается, а в файл записывается содержимое буфера для `fs2`. В итоге произошла потеря данных, в файле окажется только содержимое буфера для `fs2`.

Чтобы этого избежать, необходимо использовать флаг `O_APPEND`. Если этот флаг установлен, то первая запись в файл не теряется.

В случае многопоточной реализации потоки выполняются с разной скоростью, поэтому символы перемешаются.

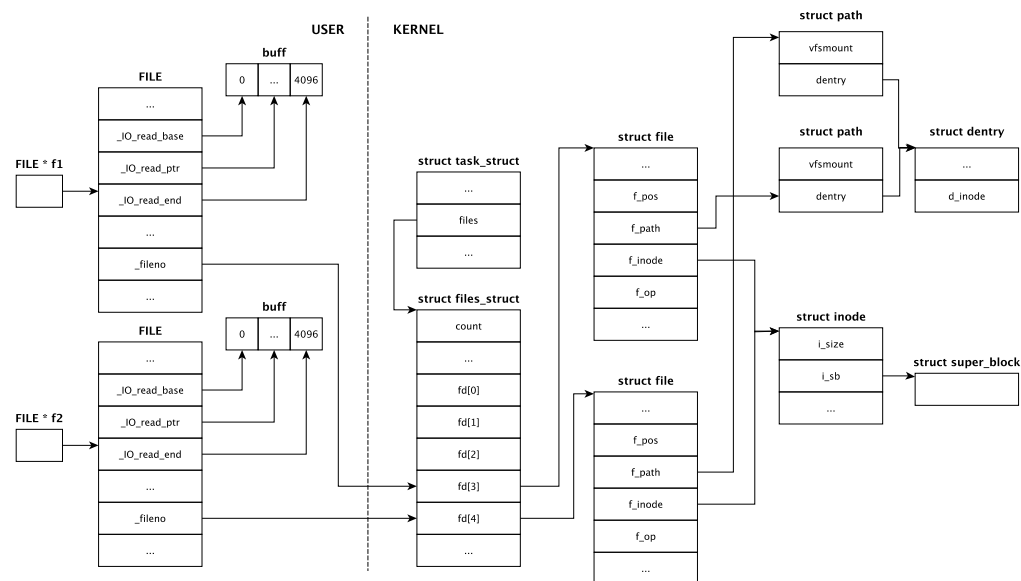


Рисунок 4.1 — Используемые структуры

## 4.2. open()

### 4.2.1. Однопоточная реализация

```

1 #include <fcntl.h>
2 #include <stdio.h>
3 #include <sys/types.h>

```



```

4 #include <sys/stat.h>
5 #include <unistd.h>
6 #define FILENAME "tmp3.txt"
7 int main()
8 {
9     struct stat buf;
10    int f1 = open(FILENAME, O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
11    int rc = fstat(f1, &buf);
12    if (!rc)
13    {
14        fprintf(stdout, "after_open_f1: inode_%ju, total_size_%lld\n", (
            uintmax_t)buf.st_ino, buf.st_size);
15    }
16    int f2 = open(FILENAME, O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
17    rc = fstat(f2, &buf);
18    if (!rc)
19    {
20        fprintf(stdout, "after_open_f2: inode_%ju, total_size_%lld\n", (
            uintmax_t)buf.st_ino, buf.st_size);
21    }
22    for (char c = 'a'; c <= 'z'; c++)
23    {
24        if (c % 2)
25        {
26            write(f1, &c, 1);
27        }
28        else
29        {
30            write(f2, &c, 1);
31        }
32    }
33    rc = fstat(f1, &buf);
34    if (!rc)
35    {
36        fprintf(stdout, "before_close_f1: inode_%ju, total_size_%lld\n", (
            uintmax_t)buf.st_ino, buf.st_size);
37    }
38    rc = fstat(f2, &buf);

```

```

39  if (!rc)
40  {
41      fprintf(stdout, "before_close_f2: inode - %ju, total_size - %lld\n", (
          uintmax_t)buf.st_ino, buf.st_size);
42  }
43  close(f1);
44  rc = stat(FILENAME, &buf);
45  fprintf(stdout, "after_close_f1: inode - %ju, total_size - %lld\n", (
          uintmax_t)buf.st_ino, buf.st_size);
46  close(f2);
47  rc = stat(FILENAME, &buf);
48  fprintf(stdout, "after_fclose_f2: inode - %ju, total_size - %lld\n", (
          uintmax_t)buf.st_ino, buf.st_size);
49  return 0;
50 }

```

### Вывод программы

```

muhomorfus@mikrovolnowka:~/Bamonka/6/os-labs/lab_08
♥ muhomorfus: ~/B/6/o/lab_08 → ./3_open_1 11:57:41
after open f1: inode - 26011861, total size - 13
after open f2: inode - 26011861, total size - 13
before close f1: inode - 26011861, total size - 13
before close f2: inode - 26011861, total size - 13
after close f1: inode - 26011861, total size - 13
after fclose f2: inode - 26011861, total size - 13
♥ muhomorfus: ~/B/6/o/lab_08 → cat tmp3.txt 11:57:47
bdfhjlnprtvx2
♥ muhomorfus: ~/B/6/o/lab_08 → | 11:57:51

```

### 4.2.2. Многопоточная реализация

```

1  #include <fcntl.h>
2  #include <pthread.h>
3  #include <stdio.h>
4  #include <unistd.h>
5  #define FILENAME "tmp4.txt"
6  void* thread1()
7  {

```

```

8  int f = open(FILENAME, O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
9  for (char c = 'a'; c <= 'z'; c += 2)
10 {
11     write(f, &c, 1);
12 }
13 close(f);
14 return NULL;
15 }
16 void* thread2()
17 {
18     int f = open(FILENAME, O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
19     for (char c = 'b'; c <= 'z'; c += 2)
20     {
21         write(f, &c, 1);
22     }
23     close(f);
24     return NULL;
25 }
26 int main()
27 {
28     pthread_t t1, t2;
29     pthread_create(&t1, NULL, thread1, NULL);
30     pthread_create(&t2, NULL, thread2, NULL);
31     pthread_join(t1, NULL);
32     pthread_join(t2, NULL);
33     return 0;
34 }

```

**Вывод программы**

```
muhomorfus@mikrovolnowka:~/Bamonka/6/os-labs/lab_08
♥ muhomorfus: ~/B/6/o/lab_08 → ./3_open_2 11:58:41
♥ muhomorfus: ~/B/6/o/lab_08 → cat tmp4.txt 11:58:44
bcegi kmoqsuwy%
♥ muhomorfus: ~/B/6/o/lab_08 → ./3_open_2 11:58:49
♥ muhomorfus: ~/B/6/o/lab_08 → cat tmp4.txt 11:58:52
acegi kmoqsuwy%
♥ muhomorfus: ~/B/6/o/lab_08 → | 11:58:53
```

### 4.2.3. Выводы

Результат работы данной программы будет таким же, как и результат предыдущей программы. При этом, причина такого поведения будет другая: при открытии файла с использованием `open()` не происходит создания буфера, таким образом, данные сразу по-символьно записываются в файл. Так как файл открыт два раза, создано два различных дескриптора открытого файла с разными `f_pos`, независимыми друг от друга.

Таким образом, при вызове функции `write()` для `f1` происходит запись символа в файл, при следующем вызове `write()` для `f2` записанный символ затирается, и на его место записывается новый символ. Таким образом, происходит потеря данных.

## 5. open() + O\_APPEND

### 5.1. Однопоточная реализация

```
1 #include <fcntl.h>
2 #include <stdio.h>
3 #include <sys/types.h>
4 #include <sys/stat.h>
5 #include <unistd.h>
6 #define FILENAME "tmp5.txt"
7 int main()
8 {
9     struct stat buf;
10    int f1 = open(FILENAME, O_WRONLY | O_CREAT | O_APPEND, S_IRUSR | S_IWUSR
        );
11    int rc = fstat(f1, &buf);
12    if (!rc)
13    {
14        fprintf(stdout, "after_open_f1: inode_%ju, total_size_%lld\n", (
            uintmax_t)buf.st_ino, buf.st_size);
15    }
16    int f2 = open(FILENAME, O_WRONLY | O_CREAT | O_APPEND, S_IRUSR | S_IWUSR
        );
17    rc = fstat(f2, &buf);
18    if (!rc)
19    {
20        fprintf(stdout, "after_open_f2: inode_%ju, total_size_%lld\n", (
            uintmax_t)buf.st_ino, buf.st_size);
21    }
22    for (char c = 'a'; c <= 'z'; c++)
23    {
24        if (c % 2)
25        {
26            write(f1, &c, 1);
27        }
28        else
```

```

29     {
30         write(f2, &c, 1);
31     }
32 }
33 rc = fstat(f1, &buf);
34 if (!rc)
35 {
36     fprintf(stdout, "before_close_f1:_inode_-%ju,_total_size_-%lld\n", (
37         uintmax_t)buf.st_ino, buf.st_size);
38 }
39 rc = fstat(f2, &buf);
40 if (!rc)
41 {
42     fprintf(stdout, "before_close_f2:_inode_-%ju,_total_size_-%lld\n", (
43         uintmax_t)buf.st_ino, buf.st_size);
44 }
45 close(f1);
46 rc = stat(FILENAME, &buf);
47 fprintf(stdout, "after_close_f1:_inode_-%ju,_total_size_-%lld\n", (
48     uintmax_t)buf.st_ino, buf.st_size);
49 close(f2);
50 rc = stat(FILENAME, &buf);
51 fprintf(stdout, "after_fclose_f2:_inode_-%ju,_total_size_-%lld\n", (
52     uintmax_t)buf.st_ino, buf.st_size);
53 return 0;
54 }

```

### Вывод программы

```
muhomorfus@mikrovolnowka:~/Bamonka/6/os-labs/lab_08
♥ muhomorfus: ~/B/6/o/Lab_08 → ./4_1 11:59:20
after open f1: inode - 26151282, total size - 0
after open f2: inode - 26151282, total size - 0
before close f1: inode - 26151282, total size - 26
before close f2: inode - 26151282, total size - 26
after close f1: inode - 26151282, total size - 26
after fclose f2: inode - 26151282, total size - 26
♥ muhomorfus: ~/B/6/o/Lab_08 → cat tmp5.txt 11:59:22
abcdefghijklmnopqrstuvwxyz%
♥ muhomorfus: ~/B/6/o/Lab_08 → | 11:59:26
```

## 5.2. Многопоточная реализация

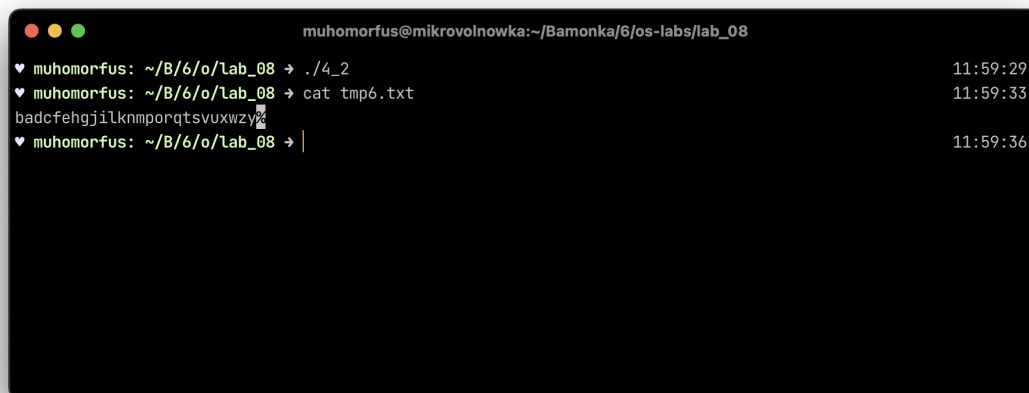
```
1 #include <fcntl.h>
2 #include <pthread.h>
3 #include <stdio.h>
4 #include <unistd.h>
5 #define FILENAME "tmp6.txt"
6 void* thread1()
7 {
8     int f = open(FILENAME, O_WRONLY | O_CREAT | O_APPEND,
9     S_IRUSR | S_IWUSR);
10    for (char c = 'a'; c <= 'z'; c += 2)
11    {
12        write(f, &c, 1);
13    }
14    close(f);
15    return NULL;
16 }
17 void* thread2()
18 {
19     int f = open(FILENAME, O_WRONLY | O_CREAT | O_APPEND,
20     S_IRUSR | S_IWUSR);
21    for (char c = 'b'; c <= 'z'; c += 2)
22    {
23        write(f, &c, 1);
```

```

24     }
25     close(f);
26     return NULL;
27 }
28 int main()
29 {
30     pthread_t t1, t2;
31     pthread_create(&t1, NULL, thread1, NULL);
32     pthread_create(&t2, NULL, thread2, NULL);
33     pthread_join(t1, NULL);
34     pthread_join(t2, NULL);
35     return 0;
36 }

```

### Вывод программы



A terminal window with a black background and white text. The title bar shows three colored circles (red, yellow, green) and the text 'muhomorfus@mikrovolnowka:~/Bamonka/6/os-labs/lab\_08'. The terminal content shows three commands being executed with their corresponding timestamps on the right:

```

♥ muhomorfus: ~/B/6/o/lab_08 → ./4_2 11:59:29
♥ muhomorfus: ~/B/6/o/lab_08 → cat tmp6.txt 11:59:33
badcfeghjilknmporqtsvuxwzy%
♥ muhomorfus: ~/B/6/o/lab_08 → | 11:59:36

```

## 5.3. Выводы

Если указан флаг `O_APPEND`, файл открывается в режиме добавления (записи в конец файла). Перед каждой операцией `write` файловый указатель будет устанавливаться в конце файла, как если бы использовался `lseek()`. Если этот флаг установлен, первая запись в файл не теряется.



## 6. Заключение

При буферизованном вводе-выводе все данные и при чтении, и при записи пишутся сначала в буфер, а только после этого в файл. При отсутствии буферизации данные сразу пишутся/читаются из файла.

В программе с `fopen()`, `fprintf()` (с буферизацией) содержимое файла затирается при вызове `fclose()`. Поэтому размер файла до вызова `fclose()` по данным из `stat()` равен нулю.

В программе с `open()`, `write()` (без буферизации) содержимое файла затирается при вызове `write()`. Поэтому размер файла до вызова `fclose()` по данным из `stat()` равен 13 — столько символов в итоге останется в файле.