



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 не буферизованный ввод-вывод

Студент Светличная А.А.

Группа ИУ7-63Б

Преподаватель Рязанова Н.Ю.

Москва — 2023 г.

1 Структура 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 area */
19    char *_IO_save_end; /* Pointer to end of non-current get area. */
20
21    struct _IO_marker *_markers;
22
23    struct _IO_FILE *_chain;
24
25    int _fileno;
26    int _flags2;
27    __off_t _old_offset; /* This used to be _offset but it's too small. */
28
29    /* 1+column number of pbase(); 0 is unknown. */
30    unsigned short _cur_column;
31    signed char _vtable_offset;
32    char _shortbuf[1];
33
34    _IO_lock_t *_lock;
35    #ifdef _IO_USE_OLD_IO_FILE
36 };
```

2 Программа №1

```
1 #include <fcntl.h>
2 #include <stdio.h>
3
4 int main() {
5     // have kernel open connection to file alphabet.txt
6     int fd = open("alphabet.txt", O_RDONLY);
7
8     // create two a C I/O buffered streams using the above connection
9     FILE *fs1 = fdopen(fd, "r");
10    char buff1[20];
11    setvbuf(fs1, buff1, _IOFBF, 20);
12
13    FILE *fs2 = fdopen(fd, "r");
14    char buff2[20];
15    setvbuf(fs2, buff2, _IOFBF, 20);
16
17    // read a char & write it alternately from fs1 and fs2
18    int flag1 = 1, flag2 = 2;
19    while (flag1 == 1 || flag2 == 1) {
20        char c;
21
22        flag1 = fscanf(fs1, "%c", &c);
23        if (flag1 == 1) fprintf(stdout, "%c", c);
24
25        flag2 = fscanf(fs2, "%c", &c);
26        if (flag2 == 1) fprintf(stdout, "%c", c);
27    }
28
29    return 0;
30 }
```



```
svetl@DESKTOP-MONLA60 MINGW64 /c/Users/svetl/Desktop/OC/LR8/code
$ ./a.exe
aubvcwdxeyfzghijklmnopqrst
```

Рисунок 2.1 – Результат работы программы

1. С помощью системного вызова `open()` создается дескриптор открытого файла (только для чтения). Системный вызов `open()` возвращает индекс в массиве `fd` структуры `files_struct`.
2. Библиотечная функция `fdopen()` возвращает указатели на `struct FILE` (`fs1` и `fs2`), которые ссылаются на дескриптор, созданный системным вызовом

`open()`.

3. Создаются буферы `buff1` и `buff2` размером 20 байт. Для дескрипторов `fs1` и `fs2` функцией `setvbuf()` задаются соответствующие буферы и тип буферизации `_IOFBF`.
4. `fscanf()` выполняется в цикле поочерёдно для `fs1` и `fs2`. При первом вызове `fscanf()` для `fs1` в буфер `buff1` считываются первые 20 символов. Значение `f_pos` в структуре `struct _file` открытого увеличится на 20. В переменную `s` записывается символ `'a'` и выводится с помощью `fprintf()`. При первом вызове `fscanf()` для `fs2` в буфер `buff2` считываются оставшиеся в файле символы (в переменную `s` записывается символ `'u'`).
5. 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.

Программа с двумя дополнительными потоками:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <fcntl.h>
4 #include <pthread.h>
5
6 void *thread_f1(void *fs)
7 {
8     char c;
9     while (fscanf((FILE *)fs, "%c", &c) == 1)
10         fprintf(stdout, "thread_1: %c\n", c);
11 }
12
13 void *thread_f2(void *fs)
14 {
15     char c;
16     while (fscanf((FILE *)fs, "%c", &c) == 1)
17         fprintf(stdout, "thread_2: %c\n", c);
18 }
19
20 int main(void)
21 {
22     int fd = open("alphabet.txt", O_RDONLY);
23     FILE *fs[2] = {fdopen(fd, "r"), fdopen(fd, "r")};
24     char buff[2][20];
25     setvbuf(fs[0], buff[0], _IOFBF, 20);
26     setvbuf(fs[1], buff[1], _IOFBF, 20);
27     char c;
28     pthread_t threads[2];
29     void *(*thread_funcs[2])(void *) = {thread_f1, thread_f2};
30
31     for (size_t i = 0; i < 2; i++)
32     {
33         if (pthread_create(&threads[i], NULL, thread_funcs[i], fs[i]) != 0)
34             exit(1);
35     }
36
37     for (size_t i = 0; i < 2; i++)
38     {
39         if (pthread_join(threads[i], NULL) != 0)
40             exit(1);
41     }
42
43     return 0;
44 }
```

```
svet1@DESKTOP-MONLA60 MINGW64 /c/Users/svet1/Desktop/OC/LR8/code
$ ./a.exe
thread 1:  a
thread 2:  u
thread 1:  b
thread 2:  v
thread 1:  c
thread 2:  w
thread 1:  d
thread 2:  x
thread 1:  e
thread 2:  y
thread 1:  f
thread 2:  z
thread 1:  g
thread 1:  h
thread 1:  i
thread 1:  j
thread 1:  k
thread 1:  l
thread 1:  m
thread 1:  n
thread 1:  o
thread 1:  p
thread 1:  q
thread 1:  r
thread 1:  s
thread 1:  t
```

Рисунок 2.2 – Результат работы 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 затрачивается время на его создание, и только потом начинается вывод. При создании дополнительных потоков связи структур не изменяются.

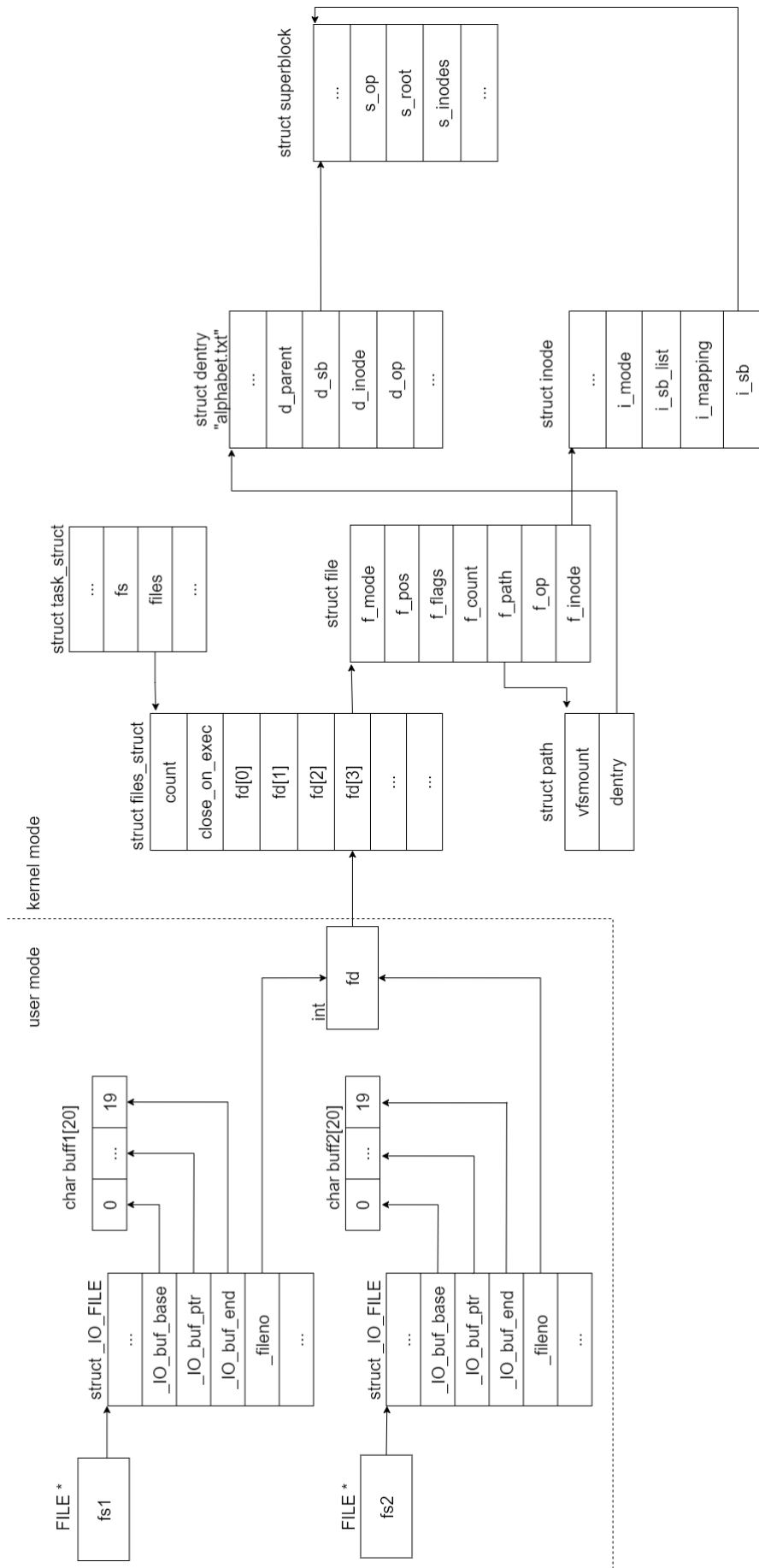
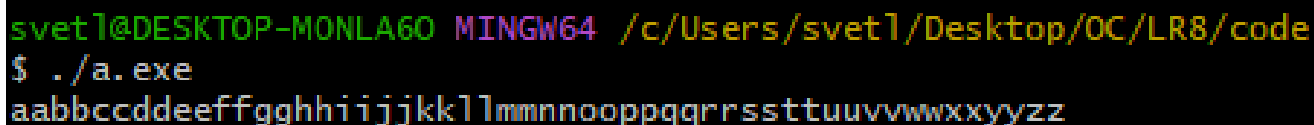


Рисунок 2.3 – Связи структур программ

3 Программа №2

```
1 #include <fcntl.h>
2 #include <unistd.h>
3
4 int main()
5 {
6     char c;
7     int fd1 = open("alphabet.txt", O_RDONLY);
8     int fd2 = open("alphabet.txt", O_RDONLY);
9
10    while (read(fd1, &c, 1) == 1 && read(fd2, &c, 1) == 1)
11    {
12        write(1, &c, 1);
13        write(1, &c, 1);
14    }
15
16    return 0;
17 }
```



```
svetl@DESKTOP-MONLA60 MINGW64 /c/Users/svetl/Desktop/OC/LR8/code
$ ./a.exe
aabbccddeeffgghhiijjkkllmmnnooppqqrrssttuuvvwwxxyyzz
```

Рисунок 3.1 – Результат работы 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 собственный 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`.

Программа с двумя дополнительными потоками:

```
1 #include <stdlib.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #include <pthread.h>
5 #include <stdio.h>
6
7 void *thread_f1(void *fd)
8 {
9     char c;
10    while (read((int)fd, &c, 1) == 1)
11        write(1, &c, 1);
12 }
13
14 void *thread_f2(void *fd)
15 {
16     char c;
17    while (read((int)fd, &c, 1) == 1)
18        write(1, &c, 1);
19 }
20
21 int main(void)
22 {
23     int fd[2] = {open("alphabet.txt", O_RDONLY),
24                  open("alphabet.txt", O_RDONLY)};
25     char c;
26     pthread_t threads[2];
27     void *(*thread_funcs[2])(void *) = {thread_f1, thread_f2};
28
29     for (size_t i = 0; i < 2; i++)
30     {
31         if (pthread_create(&threads[i], NULL, thread_funcs[i], fd[i]) != 0)
32             exit(1);
33     }
34
35     for (size_t i = 0; i < 2; i++)
36     {
37         if (pthread_join(threads[i], NULL) != 0)
38             exit(1);
39     }
40
41     return 0;
42 }
```

```

svetl@DESKTOP-MONLA60 MINGW64 /c/Users/svetl/Desktop/OC/LR8/code
$ ./a.exe
aabcdcedfegfghgihjiklmjnkolpmqnrosptqursvtwuvwxwyxyz
svetl@DESKTOP-MONLA60 MINGW64 /c/Users/svetl/Desktop/OC/LR8/code
$ ./a.exe
abcdaebfcdgheifjgkhlmjnokplqmrnsotpuqvrwsxtyzuvwxxyz

```

Рисунок 3.2 – Результат работы 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.

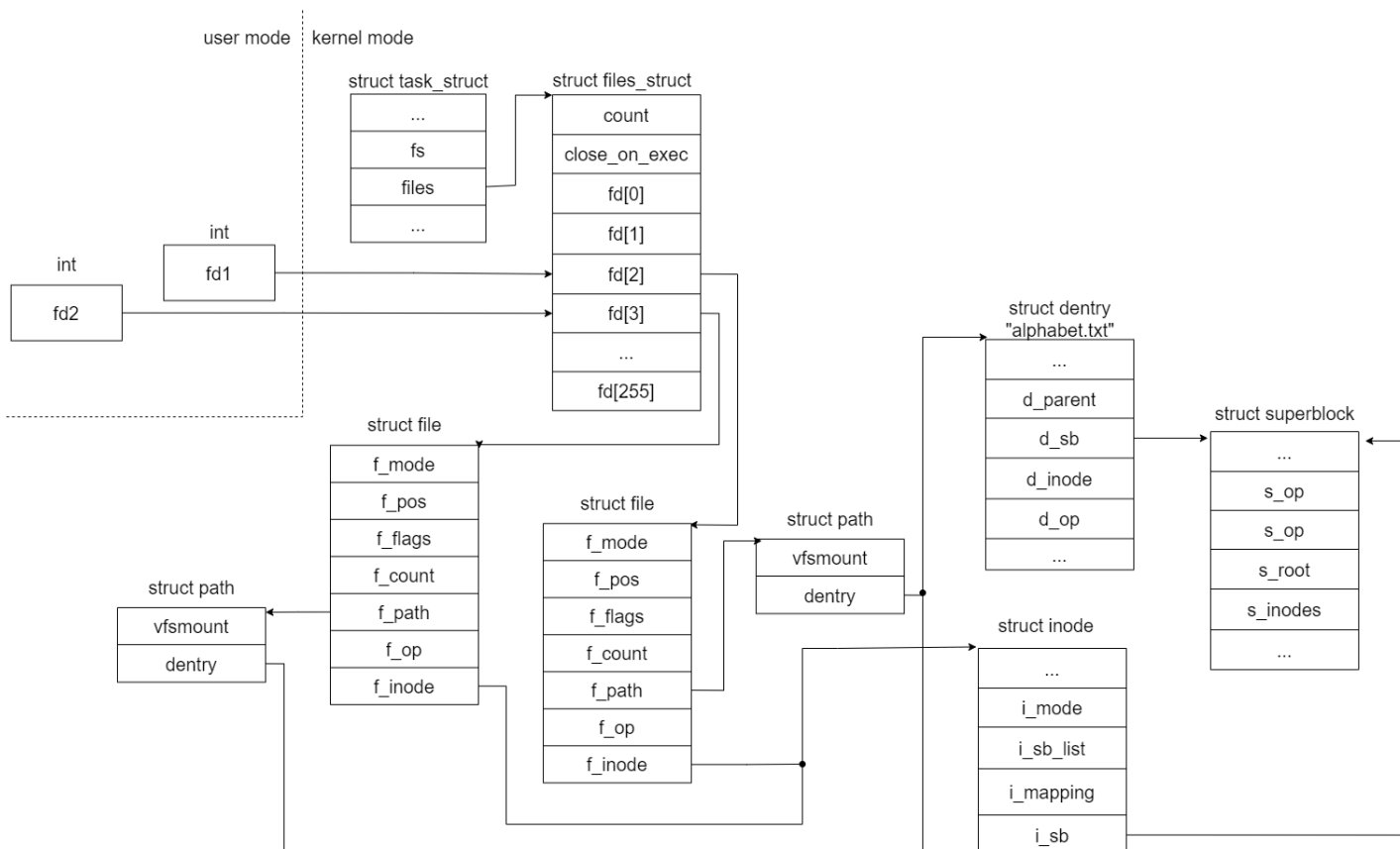


Рисунок 3.3 – Связи структур программ

4 Программа №3

Написать программу, которая открывает один и тот же файл два раза с использованием библиотечной функции `fopen()`. Для этого объявляются два файловых дескриптора. В цикле записать в файл буквы латинского алфавита поочередно передавая функции `fprintf()` то первый дескриптор, то – второй.

```
1 #include <fcntl.h>
2 #include <stdio.h>
3 #include <unistd.h>
4
5 int main()
6 {
7     FILE *f1 = fopen("out.txt", "w");
8     FILE *f2 = fopen("out.txt", "w");
9
10    for (char letr = 'a'; letr < 'z'; letr++)
11    {
12        if (letr % 2)
13            fprintf(f1, "%c", letr);
14        else
15            fprintf(f2, "%c", letr);
16    }
17
18    fclose(f2);
19    fclose(f1);
20    return 0;
21 }
```



out.txt – Блокнот

Файл Правка Формат Вид Справка

asegikmoqsuwy|

Рисунок 4.1 – Результат работы программы

Файл `out.txt` открывается функцией `fopen()` на запись дважды. Создается два дескриптора открытых файлов, две независимые позиции, но с одним и тем же `inode`. Функция `fprintf()` самостоятельно создаёт буфер, в который заносимая в файл информация первоначально и помещается. Из буфера информация переписывается в результате трех действий:

1. Информация из буфера записывается в файл когда буфер заполнен. В этом случае содержимое буфера автоматически переписывается в файл.

2. Если вызван функция `fflush` - принудительная запись содержимого в файл.
3. Если вызван функция `fclose`.

В данном случае запись в файл происходит в результате вызова функции `fclose`. При вызове `fclose()` для `f2` буфер для `f2` записывается в файл. При вызове `fclose()` для `f1`, все содержимое файла очищается, а в файл записывается содержимое буфера для `f1`. В итоге произошла утеря данных, в файле окажется только содержимое буфера для `f1`.

По этой причине необходимо использовать `open()` с флагом `O_APPEND`. Если этот флаг установлен, то каждой операции добавления гарантируется неделимость.

Программа с двумя дополнительными потоками:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <fcntl.h>
4 #include <pthread.h>
5
6 void *thread_f1(void *fs)
7 {
8     for (char c = 'a'; c <= 'z'; c += 2)
9         fprintf((FILE *)fs, "thread_1: %c\n", c);
10 }
11 void *thread_f2(void *fs)
12 {
13     for (char c = 'b'; c <= 'z'; c += 2)
14         fprintf((FILE *)fs, "thread_2: %c\n", c);
15 }
16
17 int main(void)
18 {
19     FILE *fs[2] = {fopen("s2.txt", "w"), fopen("s2.txt", "w")};
20     pthread_t threads[2];
21     void *(*thread_funcs[2])(void *) = {thread_f1, thread_f2};
22     for (size_t i = 0; i < 2; i++)
23         if (pthread_create(&threads[i], NULL, thread_funcs[i], fs[i]) != 0)
24             exit(1);
25     for (size_t i = 0; i < 2; i++)
26         if (pthread_join(threads[i], NULL) != 0)
27             exit(1);
28     fclose(fs[0]);
29     fclose(fs[1]);
30     return 0;
31 }
```

out.txt – Блокнот

Файл Правка Формат Вид Справка

```
thread 2: b
thread 2: d
thread 2: f
thread 2: h
thread 2: j
thread 2: l
thread 2: n
thread 2: p
thread 2: r
thread 2: t
thread 2: v
thread 2: x
thread 2: z
```

Рисунок 4.2 – Результат работы программы

При создании дополнительных потоков связи структур не изменяются.

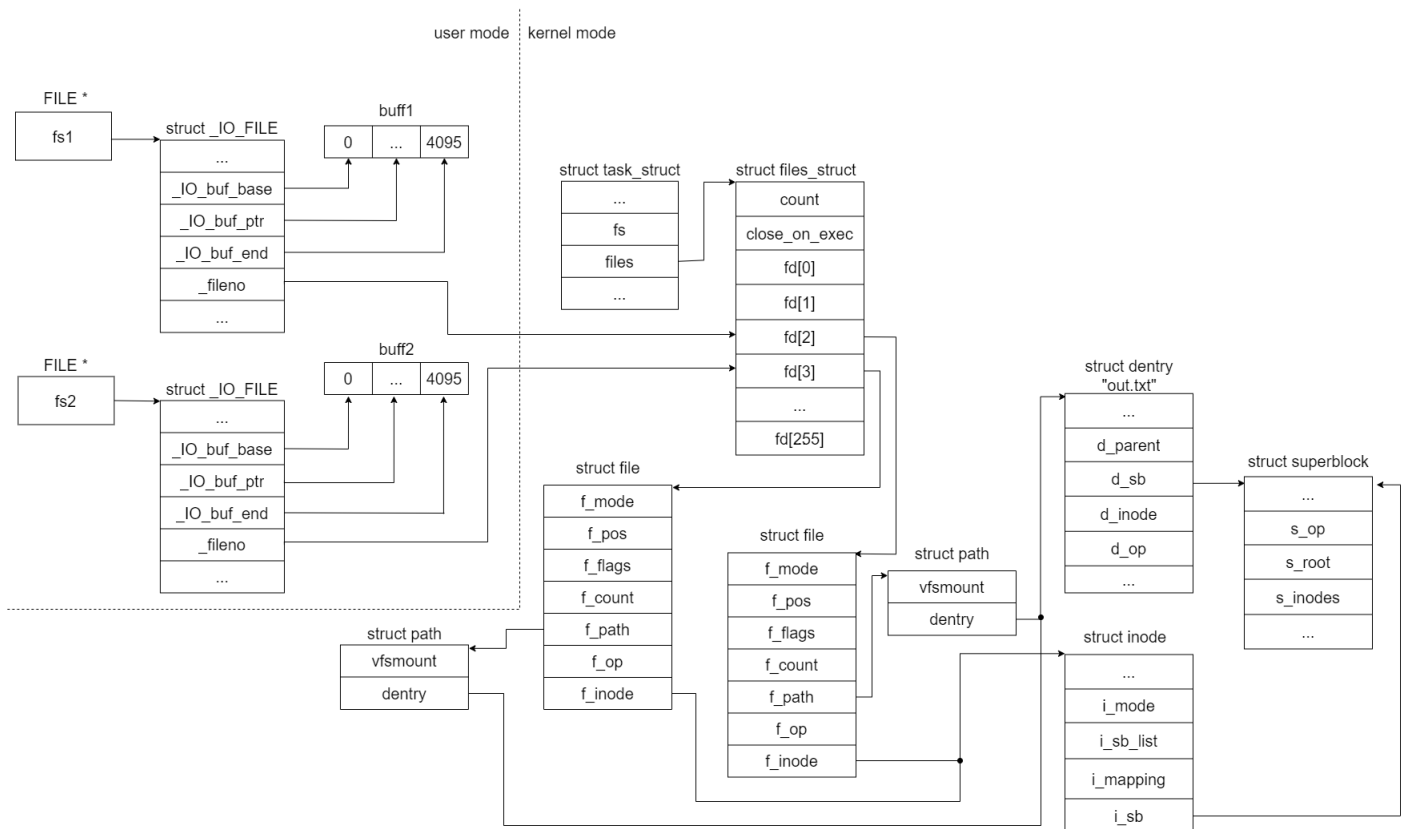


Рисунок 4.3 – Связи структур программ

5 Программа №4

```
1 #include <fcntl.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <sys/stat.h>
5 #include <unistd.h>
6
7 int main()
8 {
9     char alphabet[] = "Abcdefghijklmnopqrstuvwxyz";
10
11     int fd1 = open("out.txt", O_WRONLY | O_CREAT);
12     int fd2 = open("out.txt", O_WRONLY);
13
14     for (char i = 0; i < sizeof(alphabet); ++i)
15     {
16         if (i % 2)
17             write(fd1, alphabet + i, 1);
18         else
19             write(fd2, alphabet + i, 1);
20
21         printf("%c\n", *(alphabet + i));
22     }
23
24     close(fd1);
25     close(fd2);
26
27     return 0;
28 }
```



out.txt – Блокнот

Файл Правка Формат Вид Справка


bdfhjlnprtvxz

Рисунок 5.1 – Результат работы программы

Создаются 2 файловых дескриптора в системной таблице открытых файлов `struct _file`, для одного файла. Каждый имеет свое поле `f_pos` -> при записи в файл происходит потеря данных.

Программа с дополнительным потоком:

```
1 #include <fcntl.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <pthread.h>
5 #include <sys/stat.h>
6 #include <unistd.h>
7 #include <stdlib.h>
8
9 void *thread_f1(void *args)
10 {
11     int fd = open("out.txt", O_WRONLY | O_CREAT | O_APPEND);
12     char alphabet[] = "nopqrstuvwxyz";
13
14     for (char i = 0; i < sizeof(alphabet); ++i)
15     {
16         write(fd, alphabet + i, 1);
17         printf("%c\n", *(alphabet + i));
18     }
19
20     close(fd);
21 }
22
23 int main()
24 {
25     char alphabet[] = "abcdefghijklm";
26
27     pthread_t thread;
28     int rc = pthread_create(&thread, NULL, thread_f1, NULL);
29
30     int fd = open("out.txt", O_WRONLY | O_CREAT | O_APPEND);
31
32     for (char i = 0; i < sizeof(alphabet); ++i)
33     {
34         write(fd, alphabet + i, 1);
35         printf("%c\n", *(alphabet + i));
36     }
37
38     pthread_join(thread, NULL);
39     close(fd);
40
41     return 0;
42 }
```

 out.txt – Блокнот

Файл Правка Формат Вид Справка
abcdefghijklm

Рисунок 5.2 – Результат работы программы

При создании дополнительных потоков связи структур не изменяются.

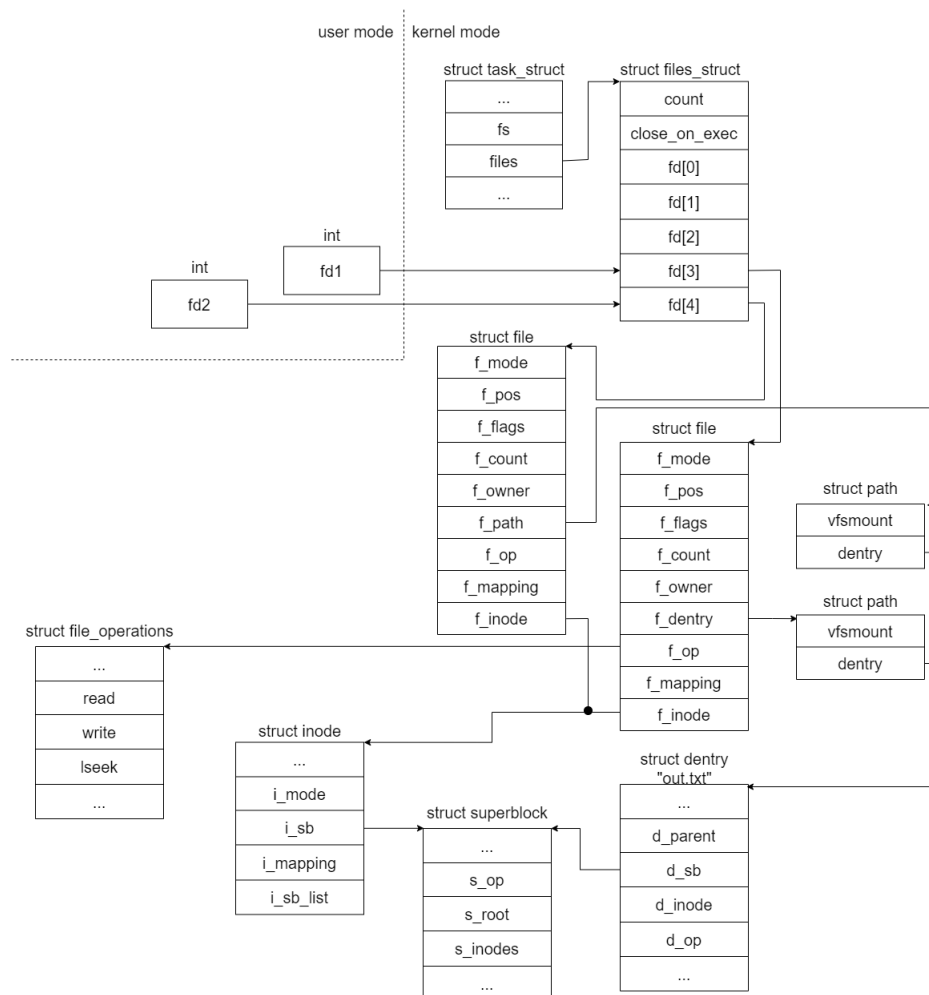


Рисунок 5.3 – Связи структур программ