



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 №5 по курсу "Операционные системы"

Тема Буферизованный и не буферизованный ввод-вывод

Студент Цветков И.А.

Группа ИУ7-63Б

Оценка (баллы) _____

Преподаватель Рязанова Н. Ю.

Москва — 2022 г.

1 Выполнение лабораторной работы

1.1 Программа 1

1.1.1 Однопоточная

Листинг 1.1 – Программа 1 (однопоточная)

```
1 #include <linux/fs.h>
2 #include <stdio.h>
3 #include <fcntl.h>
4 #include <sys/stat.h>
5 #include <unistd.h>
6
7 #define OK 0
8 #define FILE_NAME "alphabet.txt"
9 #define BUFF_SIZE 18
10
11 #define RED "\x1b[31m"
12 #define GREEN "\x1b[32m"
13 #define RESET "\x1b[0m"
14
15
16 int main()
17 {
18     int fd = open(FILE_NAME, O_RDONLY); // O_RDONLY - только на чтение.
19
20     FILE *fs1 = fdopen(fd, "r");
21     char buf1[BUFF_SIZE];
22
23     // Функция изменяет буфер, который будет использоваться для
24     // операций ввода/вывода с указанным потоком.
25     // Эта функция позволяет задать режим доступа и размер буфера.
26     setvbuf(fs1, buf1, _IOFBF, BUFF_SIZE); // _IOFBF - блочная буферизация.
27
28     FILE *fs2 = fdopen(fd, "r");
29     char buf2[BUFF_SIZE];
30     setvbuf(fs2, buf2, _IOFBF, BUFF_SIZE);
31
32     int rc1 = 1, rc2 = 1;
33
34     printf("\nResult:\n");
35
```

```

36     while (rc1 == 1 || rc2 == 1)
37     {
38         char c;
39         rc1 = fscanf(fs1, "%c", &c);
40
41         if (rc1 == 1)
42         {
43             fprintf(stdout, RED "%c" RESET, c);
44         }
45
46         rc2 = fscanf(fs2, "%c", &c);
47
48         if (rc2 == 1)
49         {
50             fprintf(stdout, GREEN "%c" RESET, c);
51         }
52     }
53
54     printf("\n\n");
55     return OK;
56 }

```

1.1.2 Обоснование

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

В функции `main` создается файловый дескриптор для открытого на чтение файла «alphabet.txt» с помощью функции `open()`. Затем при помощи `fdopen()` создаются два указателя на структуру `FILE`, в которой поле `_fileno` = 3 (дескриптор, который вернула функция `open()`). Функция `setvbuf()` изменяет тип буферизации для `fs1` и `fs2` на полную буферизацию, а также явно задает размер буфера (18 байт).

При первом вызове `fscanf()` буфер `buf1` будет заполнен полностью (все 13 символов). При этом значение `f_pos` в структуре `struct_file` открытого файла увеличится на 18. Далее при первом вызове `fscanf()` для `fs2` в `buf2` считаются оставшиеся 8 символов, начиная с `f_pos` (т.к. `fs1` и `fs2` ссылаются

на один и тот же дескриптор `fd`).

После чего в цикле поочередно будут выведены символы из `buf1` и `buf2` (при этом после 8 итераций символы будут выводиться только из `buf1`).

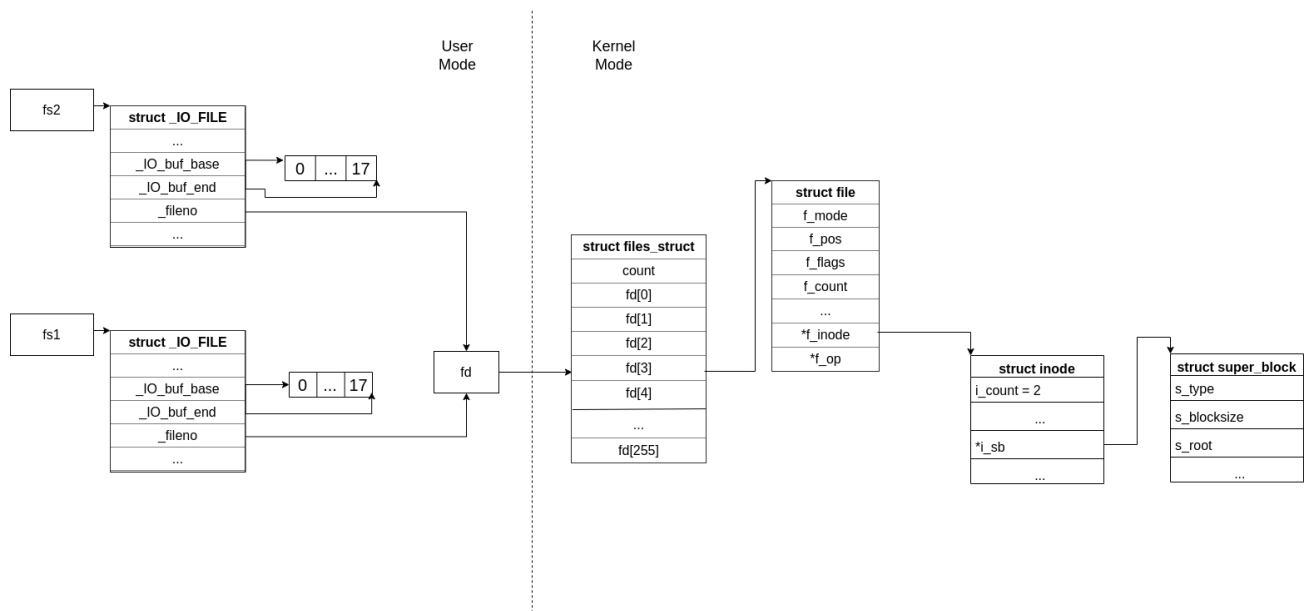


Рисунок 1.1 – Связь между структурами в первой программе

```

./appl.exe

Result:
ASBTCUDVEWFXGYHZIJKLMNOPQR

```

Рисунок 1.2 – Результат работы первой программы (однопоточная)

1.1.3 Многопоточная

Листинг 1.2 – Программа 1 (многопоточная)

```

1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <pthread.h>
4
5 #include <unistd.h> // read, write.
6
7 #define OK 0
8 #define BUF_SIZE 18
9

```

```

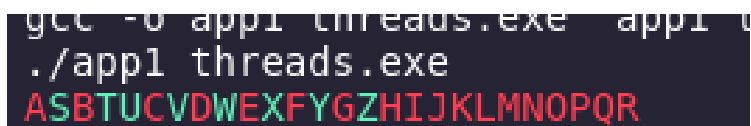
10 #define RED      "\x1b[31m"
11 #define GREEN    "\x1b[32m"
12 #define RESET    "\x1b[0m"
13
14
15 void *launchThread(void *args)
16 {
17     int rc2 = 1;
18     FILE *fs = (FILE *)args;
19
20     while (rc2 == 1)
21     {
22         char c;
23         if ((rc2 = fscanf(fs, "%c\n", &c)) == 1)
24         {
25             fprintf(stdout, GREEN "%c" RESET, c);
26             sleep(1);
27         }
28     }
29
30     return NULL;
31 }
32
33
34 int main(void)
35 {\
36     int fd = open("alphabet.txt", O_RDONLY); // O_RDONLY - только на чтение.
37
38     FILE *fs1 = fdopen(fd, "r");
39     char buf1[BUF_SIZE];
40     setvbuf(fs1, buf1, _IOFBF, BUF_SIZE); // _IOFBF - блочная буферизация.
41
42     FILE *fs2 = fdopen(fd, "r");
43     char buf2[BUF_SIZE];
44     setvbuf(fs2, buf2, _IOFBF, BUF_SIZE);
45
46     pthread_t thread;
47     int rc = pthread_create(&thread, NULL, launchThread, (void *)fs2);
48
49     int rc1 = 1;
50     while (rc1 == 1)
51     {
52         char c;
53         rc1 = fscanf(fs1, "%c\n", &c);
54
55         if (rc1 == 1)
56         {
57             fprintf(stdout, RED "%c" RESET, c);

```

```

58         sleep(1);
59     }
60 }
61
62 pthread_join(thread, NULL);
63
64 printf("\n\n");
65
66 return OK;
67 }

```



```

gcc -o app1 threads.exe app1.c
./app1 threads.exe
ASBTUCVDWEXFYGZHIJKLMNOPQR

```

Рисунок 1.3 – Результат работы первой программы (многопоточная)

1.2 Программа 2

1.2.1 Однопоточная

Листинг 1.3 – Программа 2 (однопоточная)

```

1 #include <fcntl.h>
2 #include <unistd.h> // read, write.
3
4
5 // write записывает до count байтов из буфера buf в файл,
6 // на который ссылается файловый дескриптор fd.
7 int main()
8 {
9     char c;
10
11     int fd1 = open("alphabet.txt", O_RDONLY);
12     int fd2 = open("alphabet.txt", O_RDONLY);
13     int rc1, rc2 = 1;
14
15     while (rc1 == 1 || rc2 == 1)
16     {
17         char c;
18

```

```

19     rc1 = read(fd1, &c, 1);
20     if (rc1 == 1)
21     {
22         write(1, &c, 1);
23     }
24
25     rc2 = read(fd2, &c, 1);
26     if (rc2 == 1)
27     {
28         write(1, &c, 1);
29     }
30 }
31
32 write(1, "\n", 1);
33 return 0;
34 }

```

1.2.2 Обоснование

В этой программе создаются два дескриптора открытого файла при помощи функции `open()`. При этом в системной таблице открытых файлов создаются две новых записи. Затем в цикле поочередно считываются символы из файла и выводятся на экран (при этом на экран символы будут выводиться дважды, так как каждый `open()` создаст структуру `struct file`, каждая из которых будет иметь свой `f_pos`, поэтому смещения в файловых дескрипторах будут независимы).

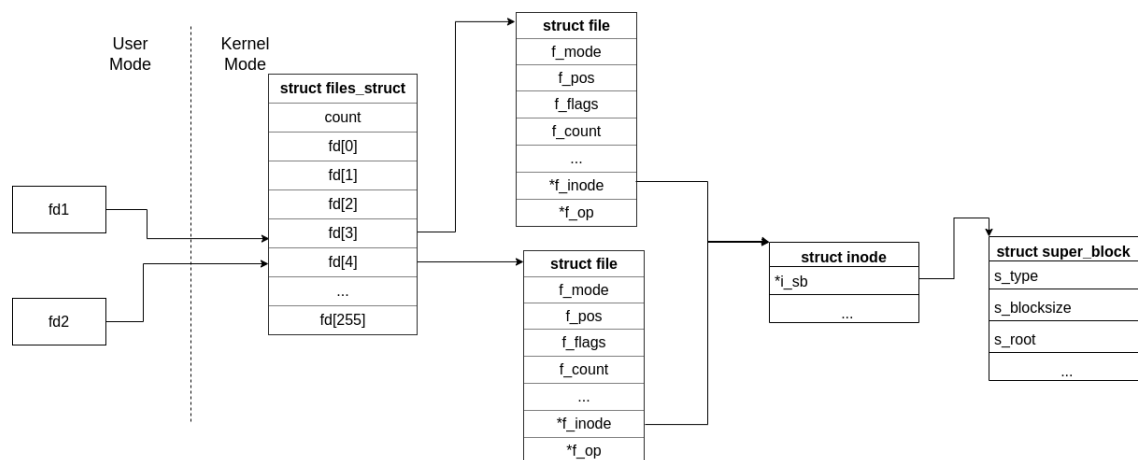


Рисунок 1.4 – Связь между структурами во второй программе

```
gcc -o app2.exe app2.o -pthread
./app2.exe
AABBCCDDEEFFGGHHIIJJKKLLMMNN00PPQRRSSTTUUVVWWXXYYZZ
amunra23@amunra23:~/studying/sem6/os/os_github/lab_05/src$
```

Рисунок 1.5 – Результат работы второй программы (однопоточная)

1.2.3 Многопоточная

Листинг 1.4 – Программа 2 (многопоточная)

```
1 #include <fcntl.h>
2 #include <unistd.h> // read, write.
3 #include <pthread.h>
4 #include <stdio.h>
5
6 #define RED      "\x1b[31m"
7 #define GREEN    "\x1b[32m"
8 #define RESET    "\x1b[0m"
9
10 pthread_mutex_t mutex;
11
12
13 void *thr_fn(void *arg)
14 {
15     int fd = open("alphabet.txt", O_RDONLY);
16     int rc = 1;
17     char c;
18
19     pthread_mutex_lock(&mutex);
20     while ((rc = read(fd, &c, 1)) == 1)
21     {
22         write(1, &c, 1);
23     }
24     pthread_mutex_unlock(&mutex);
25 }
26
27 int main()
28 {
29     pthread_t tid;
30     int rc = 1;
31     char c;
32
33     int fd = open("alphabet.txt", O_RDONLY);
34
35     int err = pthread_create(&tid, NULL, thr_fn, 0);
36     if (err)
```



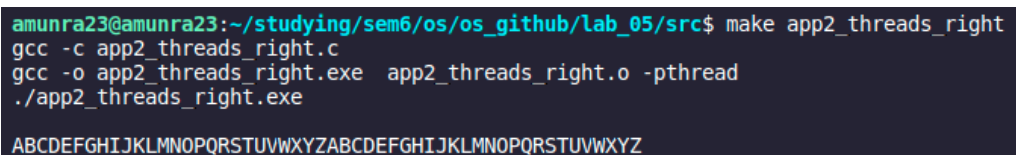
```

37 {
38     printf("Unable to create a thread");
39     return -1;
40 }
41
42 write(1, "\n", 1);
43
44 pthread_mutex_lock(&mutex);
45 while ((rc = read(fd, &c, 1)) == 1)
46 {
47     write(1, &c, 1);
48 }
49 pthread_mutex_unlock(&mutex);
50
51 pthread_join(tid, NULL);
52
53 write(1, "\n", 1);
54
55 return 0;
56 }

```

1.2.4 Обоснование

TODO



```

amunra23@amunra23:~/studying/sem6/os/os_github/lab_05/src$ make app2_threads_right
gcc -c app2_threads_right.c
gcc -o app2_threads_right.exe app2_threads_right.o -pthread
./app2_threads_right.exe
ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ

```

Рисунок 1.6 – Результат работы второй программы (многопоточная)

1.3 Программа 3

1.3.1 Однопоточная

Листинг 1.5 – Программа 3 (однопоточная)

```

1 #include <stdio.h>
2 #include <fcntl.h>

```

```

3 #include <unistd.h>
4 #include <sys/stat.h>
5
6 #define FILE_OUT "res_app3.txt"
7
8
9 void getInfo(long fPos)
10 {
11     struct stat statbuf;
12
13     stat(FILE_OUT, &statbuf);
14     printf("inode: %ld\n", statbuf.st_ino);
15     printf("Общий размер в байтах: %ld\n", statbuf.st_size);
16     printf("Размер блока ввода-вывода: %ld\n", statbuf.st_blksize);
17     printf("Текущая позиция: %ld\n\n", fPos);
18 }
19
20
21 int main()
22 {
23     long fPos;
24
25     FILE *f1 = fopen(FILE_OUT, "w");
26     fPos = ftell(f1);
27     getInfo(fPos);
28
29     FILE *f2 = fopen(FILE_OUT, "w");
30     fPos = ftell(f2);
31     getInfo(fPos);
32
33     for (char c = 'A'; c <= 'Z'; c++)
34     {
35         if (c % 2)
36         {
37             fprintf(f1, "%c", c);
38         }
39         else
40         {
41             fprintf(f2, "%c", c);
42         }
43     }
44
45     fPos = ftell(f1);
46     fclose(f1);
47     getInfo(fPos);
48
49     fPos = ftell(f2);
50     fclose(f2);

```

```

51     getInfo(fPos);
52
53     return 0;
54 }

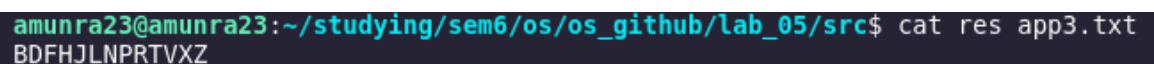
```

1.3.2 Обоснование

В этой программе файл открывается 2 раза для записи. Ввод при этом выполняется через функцию буферизированного вывода `fprintf()` стандартной библиотеки `stdio.h` языка C (буфер создается без явного вмешательства). Изначально информация записывается в буфер, а затем переписывается в файл, если буфер полон, произошла принудительная запись функцией `fflush()` или если вызван `fclose`.

В программе буквы алфавита, которые имеют нечетный код в таблице ASCII записываются в буфер дескриптора `f1`, а четные — в буфер дескриптора `f2`. Информация из буфера будет записана в файл при вызове `fclose()`. Но поскольку у каждого из дескрипторов свое поле `f_pos`, то запись будет производиться с начала файла при вызове `fclose()` для `f1` и `f2`. В результате информация будет перезаписана при втором вызове `fclose()`.

Причем если сначала был вызван `fclose(f1)`, а затем `fclose(f2)`, то в файл будут записаны *четные* буквы английского алфавита (рис. 1.7), а если `fclose()` будут вызваны в обратном порядке, то в файл запишутся *нечетные* буквы (рис. 1.8).

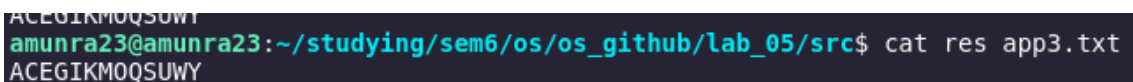


```

amunra23@amunra23:~/studying/sem6/os/os_github/lab_05/src$ cat res app3.txt
BDFHJLNPRTVXZ

```

Рисунок 1.7 – Результат при вызове `fclose()` для `f1`, затем `f2`



```

amunra23@amunra23:~/studying/sem6/os/os_github/lab_05/src$ cat res app3.txt
ACEGIKMQSUWY

```

Рисунок 1.8 – Результат при вызове `fclose()` для `f2`, затем `f1`

1.3.3 Многопоточная

```

./app3.exe
inode: 3302839
Общий размер в байтах: 0
Размер блока ввода-вывода: 4096
Текущая позиция: 0

inode: 3302839
Общий размер в байтах: 0
Размер блока ввода-вывода: 4096
Текущая позиция: 0

inode: 3302839
Общий размер в байтах: 13
Размер блока ввода-вывода: 4096
Текущая позиция: 13

inode: 3302839
Общий размер в байтах: 13
Размер блока ввода-вывода: 4096
Текущая позиция: 13

```

Рисунок 1.9 – Результат работы третьей программы - информация (однопоточная)

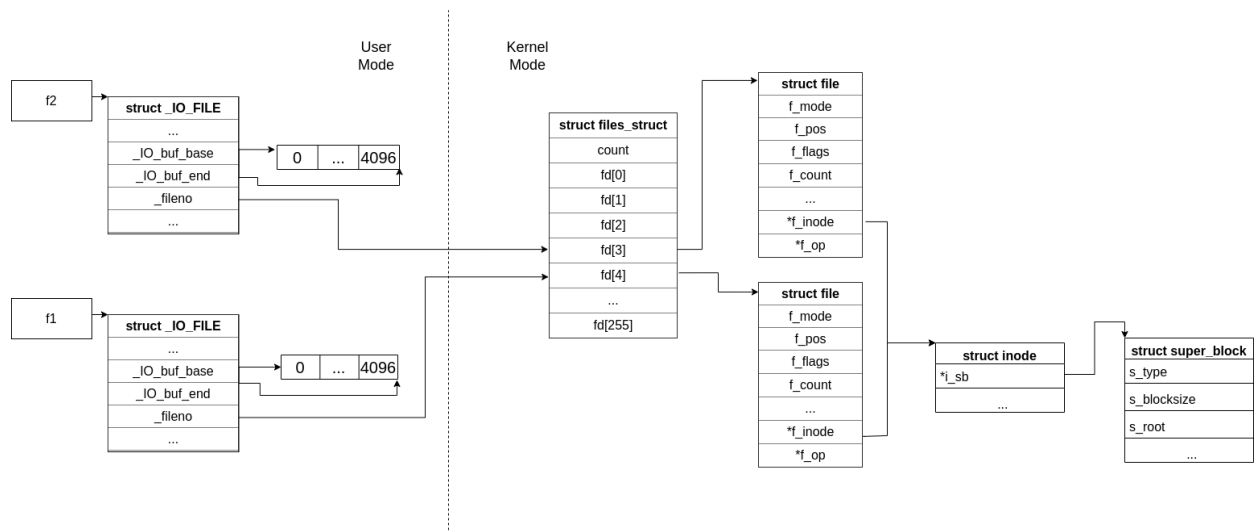


Рисунок 1.10 – Связь между структурами в третьей программе

Листинг 1.6 – Программа 3 (многопоточная)

```

1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <pthread.h>
4 #include <unistd.h>
5 #include <sys/stat.h>
6
7 #define FILE_OUT "res_app3_threads.txt"
8
9
10 void getInfo(long fPos)
11 {
12     struct stat statbuf;
13
14     stat(FILE_OUT, &statbuf);

```

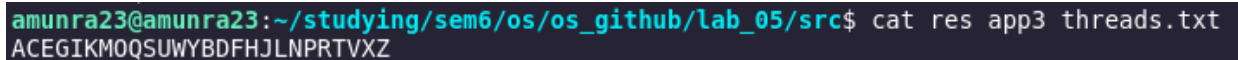
```

15     printf("inode: %ld\n", statbuf.st_ino);
16     printf("Общий размер в байтах: %ld\n", statbuf.st_size);
17     printf("Размер блока ввода-вывода: %ld\n", statbuf.st_blksize);
18     printf("Текущая позиция: %ld\n\n", fPos);
19 }
20
21 void writeChar(char c)
22 {
23     long fPos;
24     FILE *f = fopen(FILE_OUT, "a");
25     fPos = ftell(f);
26     getInfo(fPos);
27
28     while (c <= 'Z')
29     {
30         fprintf(f, "%c", c);
31         c += 2;
32     }
33
34     fPos = ftell(f);
35     fclose(f);
36     getInfo(fPos);
37 }
38
39 void *launchThread(void *arg)
40 {
41     writeChar('A');
42 }
43
44 int main()
45 {
46     pthread_t thread;
47     int rc = pthread_create(&thread, NULL, launchThread, NULL);
48
49     if (rc)
50     {
51         printf("Unable to create a thread");
52         return -1;
53     }
54
55     writeChar('B');
56
57     pthread_join(thread, NULL);
58     return 0;
59 }

```

1.3.4 Обоснование

При многопоточной реализации возникает та же проблема в перезаписью. В данной реализации в качестве решения проблемы предложено открывать файл с опцией «a» (O_APPEND) (тогда операция записи в файл станет атомарной, и перед каждым вызовом функции записи в файл смещение в файле будет устанавливаться в конец файла). При этом информация не будет перезаписана, а буферы будут в порядке вызова `fclose()` записаны в файл.

A terminal window with a dark background. The prompt is 'amunra23@amunra23:~/studying/sem6/os/os_github/lab_05/src\$'. The command 'cat res app3 threads.txt' has been executed, and the output is 'ACEGIKMQSUWYBDFHJLNPRTVXZ' on the next line.

```
amunra23@amunra23:~/studying/sem6/os/os_github/lab_05/src$ cat res app3 threads.txt
ACEGIKMQSUWYBDFHJLNPRTVXZ
```

Рисунок 1.11 – Результат работы третьей программы (многопоточная)

Вывод

При работе с буферизированным вводом-выводом может возникнуть ряд проблем, которые демонстрируются в программах данной лабораторной работы.

В первой программе использование буферизации приводит к тому, что символы выводятся не последовательно (вывод символов происходит поочередно из двух буферов, из-за чего вывод «перемешан»).

Во второй программе возникает проблема двойного вывода символов на экран, так как из-за двух файловых дескрипторов смещения в файле будут производиться независимо. Решением данной проблемы является использование `mutex` и создание разделяемой области памяти (некой переменной, которая будет отслеживать позицию указателя в файле).

В третьей программе демонстрируется проблема перезаписи информации при вызове `fclose()` для разных файловых потоков, так как каждый начинает запись с начала файла. Решением проблемы является открытие файла на дозапись с опцией «a» (`O_APPEND`) (тогда операция записи в файл станет атомарной, и перед каждым вызовом функции записи в файл смещение в файле будет устанавливаться в конец файла).

Дополнение

Структура FILE

В файле «/usr/include/x86_64-linux-gnu/bits/types/FILE.h» описание структуры FILE.

```
1 typedef struct _IO_FILE FILE;
```

В файле «/usr/include/x86_64-linux-gnu/bits/libio.h» находится описание структуры _IO_FILE.

```
1 struct _IO_FILE
2 {
3     int _flags; /* High-order word is _IO_MAGIC; rest is flags. */
4     #define _IO_file_flags _flags
5
6     /* The following pointers correspond to the C++ streambuf protocol. */
7     /* Note: Tk uses the _IO_read_ptr and _IO_read_end fields directly. */
8     char *_IO_read_ptr; /* Current read pointer */
9     char *_IO_read_end; /* End of get area. */
10    char *_IO_read_base; /* Start of putback+get area. */
11    char *_IO_write_base; /* Start of put area. */
12    char *_IO_write_ptr; /* Current put pointer. */
13    char *_IO_write_end; /* End of put area. */
14    char *_IO_buf_base; /* Start of reserve area. */
15    char *_IO_buf_end; /* End of reserve area. */
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    #if 0
27    int _blksize;
28    #else
29    int _flags2;
30    #endif
31    _IO_off_t _old_offset; /* This used to be _offset but it's too small. */
32
33    #define __HAVE_COLUMN /* temporary */
34    /* 1+column number of pbase(); 0 is unknown. */
35    unsigned short _cur_column;
36    signed char _vtable_offset;
```



```
37 char _shortbuf[1];
38
39 /* char* _save_gp_ptr; char* _save_eg_ptr; */
40
41 _IO_lock_t *_lock;
42 #ifdef _IO_USE_OLD_IO_FILE
43 };
```