



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 дисциплине «Операционные системы»

Тема Буферизованный ввод / вывод

Студент Романов А.В.

Группа ИУ7-63Б

Оценка (баллы) _____

Преподаватель Рязанова Н. Ю.

1 | Структура `_IO_FILE`

Листинг 1.1: Листинг структуры `_IO_FILE`

```
1 struct _IO_FILE
2 {
3     int _flags;      /* High-order word is _IO_MAGIC; rest is flags. */
4
5     /* The following pointers correspond to the C++ streambuf protocol. */
6     char *_IO_read_ptr; /* Current read pointer */
7     char *_IO_read_end; /* End of get area. */
8     char *_IO_read_base; /* Start of putback+get area. */
9     char *_IO_write_base; /* Start of put area. */
10    char *_IO_write_ptr; /* Current put pointer. */
11    char *_IO_write_end; /* End of put area. */
12    char *_IO_buf_base; /* Start of reserve area. */
13    char *_IO_buf_end; /* End of reserve area. */
14
15    /* The following fields are used to support backing up and undo. */
16    char *_IO_save_base; /* Pointer to start of non-current get area. */
17    char *_IO_backup_base; /* Pointer to first valid character of backup area
18    */
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    };
37
```

```

37 struct _IO_FILE_complete
38 {
39     struct _IO_FILE _file;
40 #endif
41     __off64_t _offset;
42     /* Wide character stream stuff. */
43     struct _IO_codecvt *_codecvt;
44     struct _IO_wide_data *_wide_data;
45     struct _IO_FILE *_freeres_list;
46     void *_freeres_buf;
47     size_t __pad5;
48     int _mode;
49     /* Make sure we don't get into trouble again. */
50     char _unused2[15 * sizeof (int) - 4 * sizeof (void *) - sizeof (size_t)];
51 };

```

2 | Первая программа

На рис. 2.1 представлена схема структур, используемых в первой программе.

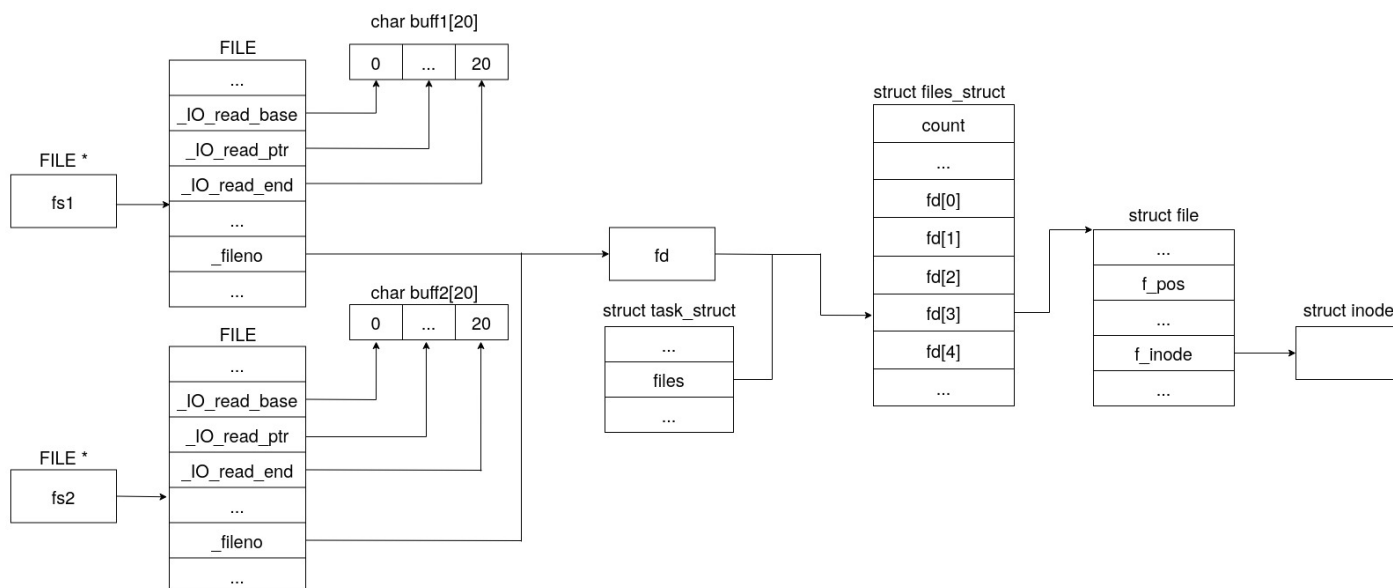


Рис. 2.1: Схема структур программы №1

- Функция `open()` создает новый файловый дескриптор файла (открытого только на чтение) "alphabet.txt" запись в системной таблице открыт файлов. Эта запись регистрирует смещение в файле и флаги состояния файла;
- функция `fdopen()` создает указатели на структуру `FILE`. Поле `_fileno` содержит дескриптор, который вернула функция `fopen()`;
- функция `setvbuf()` явно задает размер буфера в 20 байт и меняет тип буферизации (для `fs1` и `fs2`) на полную;
- при первом вызове функции `fscanf()` в цикле (для `fs1`), `buff1` будет заполнен полностью – первыми 20 символами (буквами 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`);
- в цикле поочередно выводятся символы из `buff1` и `buff2`.

Листинг 2.1: Программа №1

```
1 #include <stdio.h>
2 #include <fcntl.h>
3
4 #define OK 0
5 #define BUF_SIZE 20
6 #define VALID_READED 1
7
8 #define FILE_NAME "data/alphabet.txt"
9 #define SPEC "%c"
10
11 int main(void)
12 {
13     int fd = open(FILE_NAME, O_RDONLY);
14
15     FILE *fs1 = fdopen(fd, "r");
16     char buff1[BUF_SIZE];
17     setvbuf(fs1, buff1, _IOFBF, BUF_SIZE);
18
19     FILE *fs2 = fdopen(fd, "r");
20     char buff2[BUF_SIZE];
21     setvbuf(fs2, buff2, _IOFBF, BUF_SIZE);
22
23     int flag1 = 1, flag2 = 2;
24     while (flag1 == VALID_READED || flag2 == VALID_READED)
25     {
26         char c;
27
28         if ((flag1 = fscanf(fs1, SPEC, &c)) == VALID_READED)
29         {
30             fprintf(stdout, SPEC, c);
31         }
32
33         if ((flag2 = fscanf(fs2, SPEC, &c)) == VALID_READED)
34         {
35             fprintf(stdout, SPEC, c);
36         }
37     }
38
39     return OK;
40 }
```

На рис. 2.2 представлен результат работы первой программы.

```
alexey@alexey ~/repos/sem_06/os/sem_02/lab_05 ↵ master ± ./prog_01
AUBVCWDXEYFZGHIJKLMNOPQRSTX
```

Рис. 2.2: Результат работы первой программы

Листинг 2.2: Программа №1 (реализация с потоками)

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <pthread.h>
4 #define OK 0
5 #define BUF_SIZE 20
6 #define VALID_READED 1
7 #define FILE_NAME "data/alphabet.txt"
8 #define SPEC "%c"
9
10 void *run_buffer(void *args)
11 {
12     int flag = 1;
13     FILE *fs = (FILE *)args;
14     while (flag == VALID_READED)
15     {
16         char c;
17         if ((flag = fscanf(fs, SPEC, &c)) == VALID_READED)
18         {
19             fprintf(stdout, SPEC, c);
20         }
21     }
22     return NULL;
23 }
24
25 int main(void)
26 {
27     setbuf(stdout, NULL);
28     pthread_t thread;
29     int fd = open(FILE_NAME, O_RDONLY);
30
31     FILE *fs1 = fdopen(fd, "r");
32     char buff1[BUF_SIZE];
33     setvbuf(fs1, buff1, _IOFBF, BUF_SIZE);
34
35     FILE *fs2 = fdopen(fd, "r");
36     char buff2[BUF_SIZE];
37     setvbuf(fs2, buff2, _IOFBF, BUF_SIZE);
38
39     int rc = pthread_create(&thread, NULL, run_buffer, (void *)fs2);
40
41     int flag = 1;
42     while (flag == VALID_READED)
```

```

43 {
44     char c;
45     fprintf(stdout, "\nSCANF IN MAIN_1");
46     flag = fscanf(fs1, SPEC, &c);
47     fprintf(stdout, "\nSCANF IN MAIN_2");
48     if (flag == 1)
49     {
50         fprintf(stdout, SPEC, c);
51     }
52 }
53
54 pthread_join(thread, NULL);
55 return OK;
56 }

```

На рис. 2.3 представлен результат работы первой программы (с потоками).



```

alexey@alexey ~/repos/sem_06/os/sem_02/lab_05 ⌕ master ± ./prog_01_thread
thread 1: A
thread 1: B
thread 1: C
thread 1: D
thread 1: E
thread 1: F
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 2: U
thread 2: V
thread 2: W
thread 1: Q
thread 1: R
thread 2: X
thread 2: Y
thread 2: Z
thread 1: S
thread 1: T

```

Рис. 2.3: Результат работы первой программы (с потоками)

3 | Вторая программа

На рис. 3.1 представлена схема структур, используемых во второй программе.

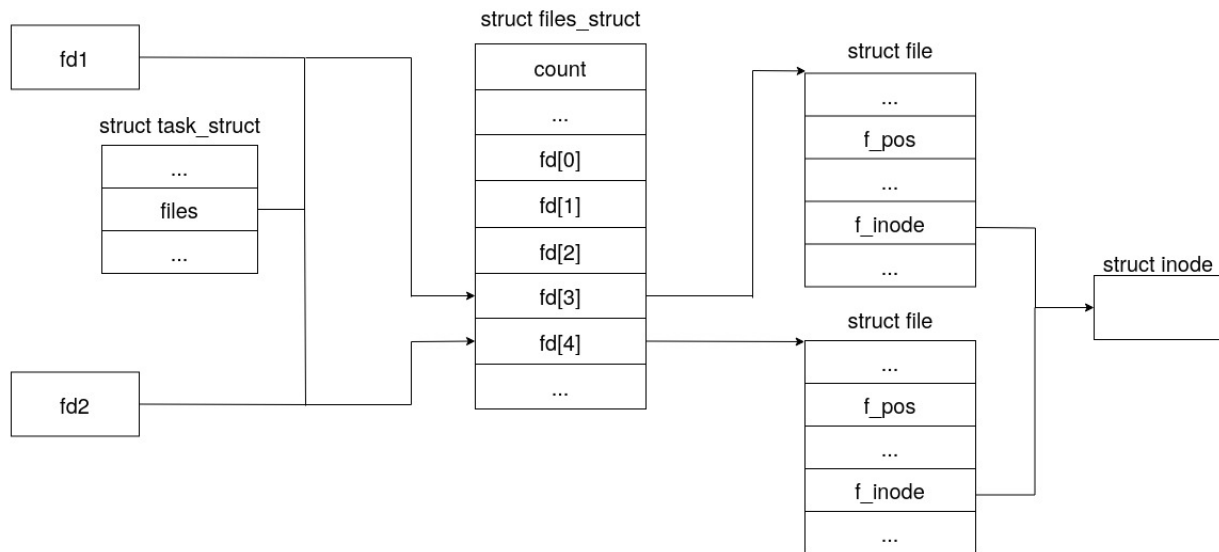


Рис. 3.1: Схема структур программы №2

- Функция `open()` создает файловые дескрипторы, два раза для одного и того же файла, поэтому в программе существует две различные `struct file`, но ссылающиеся на один и тот же `struct inode`;
- из-за того что структуры разные, посимвольная печать просто дважды выведет содержимое файла в формате «AAbbсс...» (в случае однопоточной реализации);
- в случае многопоточной реализации, вывод второго потока начнется позже (нужно время, для создание этого потока) и символы перемешаются (см. рис. 3.3).

Листинг 3.1: Программа №2

```
1 #include <fcntl.h>
2 #include <unistd.h>
3
4 #define OK 0
5 #define VALID_READED 1
6 #define FILE_NAME "data/alphabet.txt"
```



```

7
8 int main(void)
9 {
10     int fd1 = open(FILE_NAME, O_RDONLY);
11     int fd2 = open(FILE_NAME, O_RDONLY);
12     int rc1, rc2 = VALID_READED;
13     while (rc1 == VALID_READED || rc2 == VALID_READED)
14     {
15         char c;
16         rc1 = read(fd1, &c, 1);
17         if (rc1 == VALID_READED)
18         {
19             write(1, &c, 1);
20         }
21
22         rc2 = read(fd2, &c, 1);
23         if (rc2 == VALID_READED)
24         {
25             write(1, &c, 1);
26         }
27     }
28     return OK;
29 }

```

На рис. 3.2 представлен результат работы второй программы.



Рис. 3.2: Результат работы второй программы

Листинг 3.2: Программа №2 (реализация с потоками)

```

1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #include <pthread.h>
5
6 #define OK 0
7 #define VALID_READED 1
8 #define FILE_NAME "data/alphabet.txt"
9
10 void *run_buffer(void *args)
11 {
12     int fd = *((int *)args);
13     int err = VALID_READED;
14
15     while (err == VALID_READED)

```

```

16 {
17     char c;
18     err = read(fd, &c, 1);
19     if (err == VALID_READED)
20     {
21         write(1, &c, 1);
22     }
23 }
24
25 return NULL;
26 }
27
28 int main(void)
29 {
30     int fd1 = open(FILE_NAME, O_RDONLY);
31     int fd2 = open(FILE_NAME, O_RDONLY);
32
33     pthread_t thread;
34     int rc = pthread_create(&thread, NULL, run_buffer, (void *)&fd2));
35     int err = VALID_READED;
36
37     while (err == VALID_READED)
38     {
39         char c;
40         err = read(fd1, &c, 1);
41         if (err == VALID_READED)
42         {
43             write(1, &c, 1);
44         }
45     }
46
47     pthread_join(thread, NULL);
48     return OK;
49 }

```

На рис. 3.3 представлен результат работы второй программы (с потоками).



```

alexey@alexey ~/repos/sem_06/os/sem_02/lab_05 master ± ./prog_02_thread
ABCDABFCGDHEIFGJHKILJMKNLMPNQORPSQTRUSVTWUXVYWZXYZZ

```

Рис. 3.3: Результат работы второй программы (с потоками)

4 | Третья программа

На рис. 4.1 представлена схема структур, используемых в третьей программе.

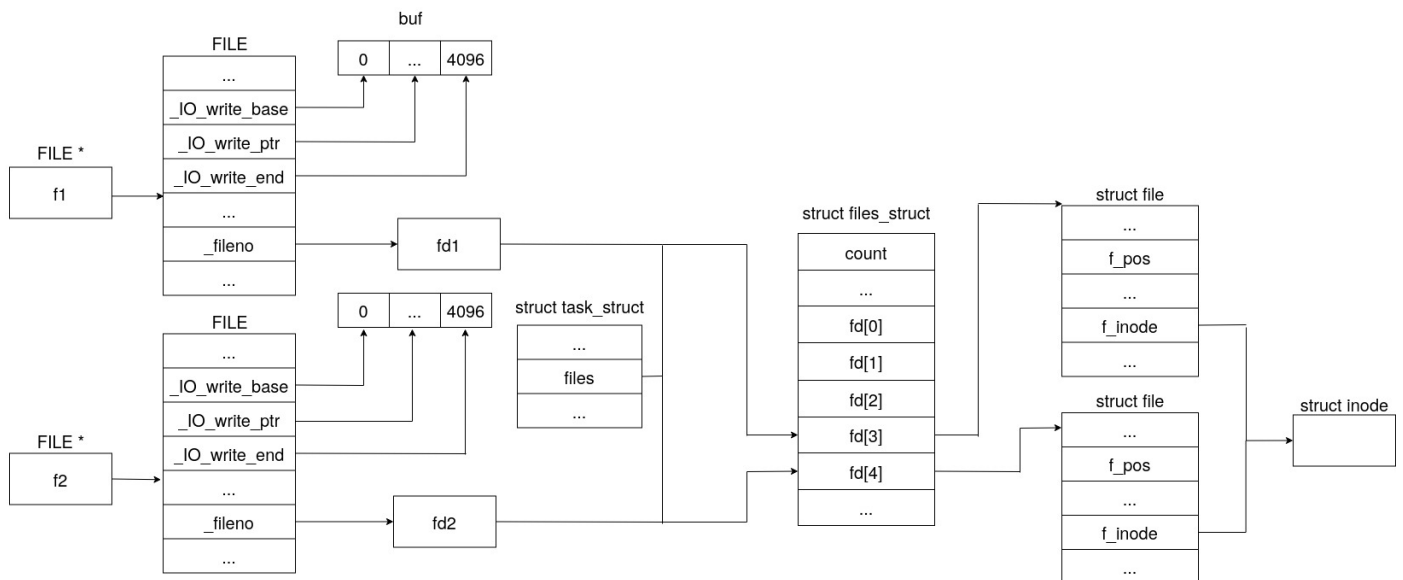


Рис. 4.1: Схема структур программы №2

- Файл открывается на запись два раза, с помощью функции `fopen()`;
- функция `fprintf()` предоставляет буферизованный вывод - буфер создается без нашего вмешательства;g
- изначально информация пишется в буфер, а из буфера в файл если произошло одно из событий:
 - буфер полон;
 - вызвана функция `fclose()`;
 - вызвана функция `fflush()`;
- в случае нашей программы, информация в файл запишется в результате вызова функция `fclose()`;
- из-за того `f_pos` независимы для каждого дескриптора файла, запись в файл будет производится с самого начала;

- таким образом, информация записанная при первом вызове `fclose()` будет потеряна в результате второго вызова `fclose()` (см. рис. 4.2).
- в многопоточной реализации результат аналогичен - с помощью `pthread_join` мы ожидаем вызова `fclose()` для `f2` в отдельном потоке и далее вызываем `fclose()` для `f1`.

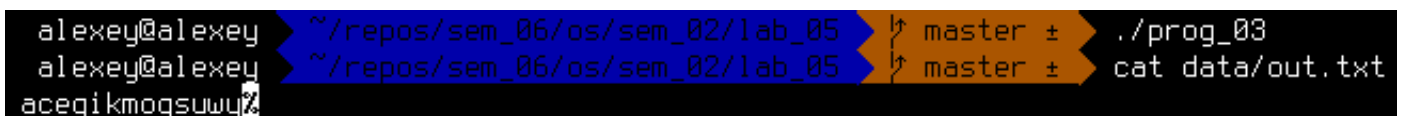
Листинг 4.1: Программа №3

```

1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #define OK 0
5 #define FILE_NAME "data/out.txt"
6 #define SPEC "%c"
7
8 int main()
9 {
10     FILE *f1 = fopen(FILE_NAME, "w");
11     FILE *f2 = fopen(FILE_NAME, "w");
12
13     for (char c = 'a'; c <= 'z'; c++)
14     {
15         if (c % 2)
16         {
17             fprintf(f1, SPEC, c);
18         }
19         else
20         {
21             fprintf(f2, SPEC, c);
22         }
23     }
24
25     fclose(f2);
26     fclose(f1);
27     return OK;
28 }

```

На рис. 4.2 представлен результат работы третьей программы.



```

alexey@alexey ~/repos/sem_06/os/sem_02/lab_05 master ± ./prog_03
alexey@alexey ~/repos/sem_06/os/sem_02/lab_05 master ± cat data/out.txt
acegikmoqsuwyz

```

Рис. 4.2: Результат работы третьей программы

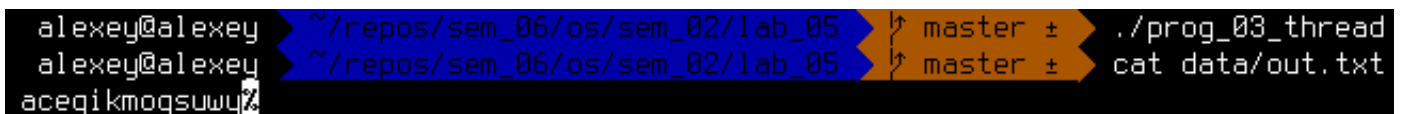
Листинг 4.2: Программа №3 (реализация с потоками)

```

1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <pthread.h>
4 #include <unistd.h>
5 #define OK 0
6 #define FILE_NAME "data/out.txt"
7 #define SPEC "%c"
8
9 void *run_buffer(void *args)
10 {
11     FILE *f = (FILE *)args;
12
13     for (char c = 'b'; c <= 'z'; c += 2)
14     {
15         fprintf(f, SPEC, c);
16     }
17
18     fclose(f);
19     return NULL;
20 }
21
22 int main()
23 {
24     FILE *f1 = fopen(FILE_NAME, "w");
25     FILE *f2 = fopen(FILE_NAME, "w");
26
27     pthread_t thread;
28     int rc = pthread_create(&thread, NULL, run_buffer, (void *)(f2));
29
30     for (char c = 'a'; c <= 'z'; c += 2)
31     {
32         fprintf(f1, SPEC, c);
33     }
34
35     pthread_join(thread, NULL);
36     fclose(f1);
37     return OK;
38 }

```

На рис. 4.3 представлен результат работы второй программы (с потоками).



```

alexey@alexey ~/repos/sem_06/os/sem_02/lab_05 ⌕ master ± ./prog_03_thread
alexey@alexey ~/repos/sem_06/os/sem_02/lab_05 ⌕ master ± cat data/out.txt
acegikmoqsuwyz

```

Рис. 4.3: Результат работы третьей программы (с потоками)