



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 небуферизованный ввод-вывод

Студент Пересторонин П.Г.

Группа ИУ7-63Б

Преподаватель Рязанова Н. Ю.

# Оглавление

<b>1</b>	<b>Первая программа</b>	<b>4</b>
1.1	Код . . . . .	4
1.1.1	Один поток . . . . .	4
1.1.2	Два потока . . . . .	4
1.2	Результат и анализ . . . . .	7
1.2.1	Связь структур в 1 программе . . . . .	7
1.2.2	С одним потоком . . . . .	7
1.2.3	С двумя потоками . . . . .	7
1.2.4	Анализ . . . . .	9
<b>2</b>	<b>Вторая программа</b>	<b>12</b>
2.1	Код . . . . .	12
2.1.1	Один поток . . . . .	12
2.1.2	Два потока . . . . .	12
2.2	Результат и анализ . . . . .	14
2.2.1	Связь структур во 2 программе . . . . .	14
2.2.2	С одним потоком . . . . .	14
2.2.3	С двумя потоками . . . . .	14
2.2.4	Анализ . . . . .	15
<b>3</b>	<b>Третья программа</b>	<b>16</b>
3.1	Код . . . . .	16
3.1.1	Один поток . . . . .	16
3.1.2	Два потока . . . . .	16
3.2	Результат и анализ . . . . .	18
3.2.1	Связь структур в 3 программе . . . . .	18
3.2.2	С одним потоком . . . . .	18
3.2.3	С двумя потоками . . . . .	18
3.2.4	Анализ . . . . .	19

# Структура FILE

Структура FILE у меня в системе описана в файле  
/usr/include/bits/types/FILE.h:

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

Структура \_IO\_FILE описана в файле  
/usr/include/bits/types/struct\_FILE.h:

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

# 1 Первая программа

## 1.1 Код

### 1.1.1 Один поток

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

Листинг 1.1: Первая программа с одним потоком

### 1.1.2 Два потока

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

```

3 #include <pthread.h>
4
5 #define BUF_SIZE 20
6 #define FILENAME "alphabet.txt"
7
8 void *run(void *arg) {
9     /*printf("\n=== Separate thread start ===\n");*/
10    FILE *fs = arg;
11    int flag = 1;
12    char c;
13
14    while (flag == 1) {
15        flag = fscanf(fs, "%c", &c);
16        if (flag == 1)
17            fprintf(stdout, "t2: %c\n", c);
18    }
19    /*printf("\n=== Separate thread end ===\n");*/
20    return NULL;
21 }
22
23 int main() {
24     /*setbuf(stdout, NULL);*/
25     int fd = open(FILENAME, O_RDONLY);
26     pthread_t td;
27
28     FILE* fs1 = fdopen(fd, "r");
29     char buff1[BUF_SIZE];
30     setvbuf(fs1, buff1, _IOFBF, BUF_SIZE);
31
32     FILE* fs2 = fdopen(fd, "r");
33     char buff2[BUF_SIZE];
34     setvbuf(fs2, buff2, _IOFBF, BUF_SIZE);
35
36     pthread_create(&td, NULL, run, fs2);
37
38     /*printf("\n=== Main thread start ===\n");*/
39
40     int flag = 1;
41     char c;
42     while (flag == 1) {
43         flag = fscanf(fs1, "%c", &c);
44         if (flag == 1)
45             fprintf(stdout, "t1: %c\n", c);
46     }
47
48     /*printf("\n=== Main thread end ===\n");*/
49
50     pthread_join(td, NULL);

```

```
51     return 0;  
52 }
```

Листинг 1.2: Первая программа с двумя потоками

## 1.2 Результат и анализ

### 1.2.1 Связь структур в 1 программе

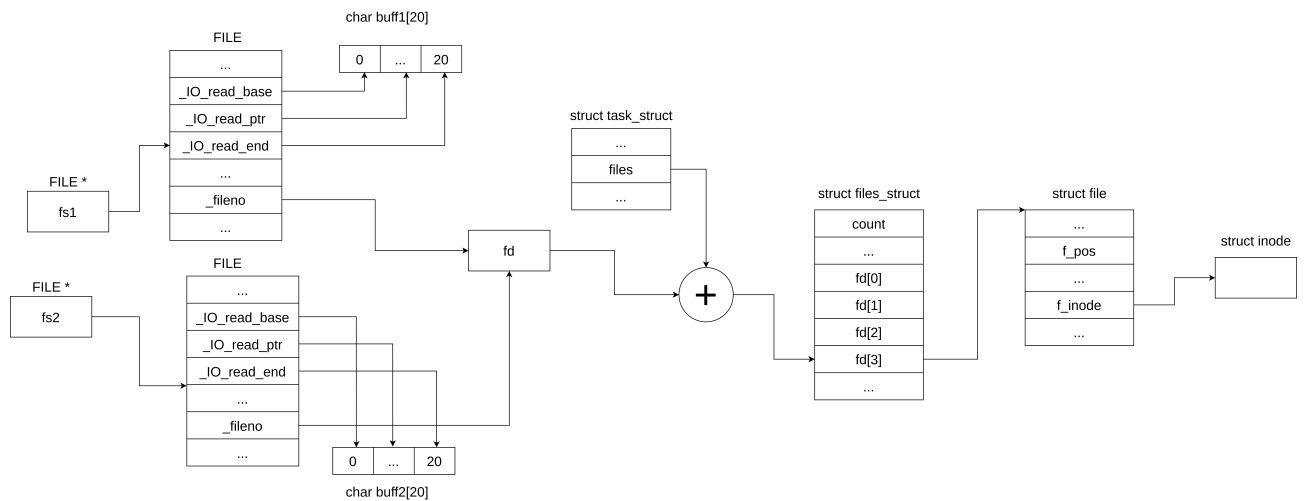


Рис. 1.1: Связь структур данных в 1 программе

### 1.2.2 С одним потоком

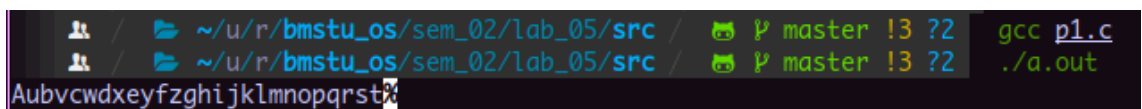


Рис. 1.2: Результат работы 1 программы с 1 потоком

### 1.2.3 С двумя потоками



```

  / ~/u/r/bmstu_os/sem_02/lab_05/src /  master !3 ?2 gcc p1mult.c -pthread
  / ~/u/r/bmstu_os/sem_02/lab_05/src /  master !3 ?2 ./a.out
t1: A
t1: b
t1: c
t1: d
t1: e
t1: f
t1: g
t1: h
t1: i
t1: j
t1: k
t1: l
t1: m
t1: n
t1: o
t1: p
t2: u
t2: v
t1: q
t1: r
t1: s
t2: w
t2: x
t2: y
t2: z
t1: t
  / ~/u/r/bmstu_os/sem_02/lab_05/src /  master !3 ?2
```

Рис. 1.3: Результат работы 1 программы с 2 потоками

## 1.2.4 Анализ

Изначально в данной программе с помощью системного вызова `open()` создается файловый дескриптор `fd` для файла `alphabet.txt` с правами доступа на чтение (`O_RDONLY`), этому файловому дескриптору присваивается значение 3 (потому что 0, 1, 2 заняты `stdin`, `stdout` и `stderr` соответственно). В это же время у `struct files_struct` (можно найти в `struct task_struct` для текущего процесса (структура ядра), поле `files`) поле `fd[3]` начинает указывать на `struct file`, связанный с `struct inode`, соответствующий файлу с именем `alphabet.txt`.

Далее, с помощью вызова функции стандартной библиотеки `fdopen()`, создаются 2 структуры `FILE` (`fs1`, `fs2`), поле `_fileno` становится равным значению 3.

Затем с помощью вызова функции `setvbuf` создаются буферы для обеих структур `FILE`, в качестве аргумента функция получает буфер, размер буфера, а также стратегию буферизации (по строчкам (то есть до ближайшего символа `”`) или полностью заполняя буфер). При установке буфера в структуре меняются значения полей, отвечающих за буфер (см. рисунок 1.4).

```
(gdb) p *fs1
$2 = {_flags = -72539000, _IO_read_ptr = 0x0, _IO_read_end = 0x0, _IO_read_base = 0x0, _IO_write_base = 0x0,
      _IO_write_ptr = 0x0, _IO_write_end = 0x0, _IO_buf_base = 0x0, _IO_buf_end = 0x0, _IO_save_base = 0x0,
      _IO_backup_base = 0x0, _IO_save_end = 0x0, _markers = 0x0, _chain = 0x7ffff7f89440 <_IO_2_1_stderr_>,
      _fileno = 3, _flags2 = 0, _old_offset = 0, _cur_column = 0, _vtable_offset = 0 '\000', _shortbuf = "",
      _lock = 0x55555559380, _offset = -1, _codecvt = 0x0, _wide_data = 0x55555559390, _freeres_list = 0x0,
      _freeres_buf = 0x0, __pad5 = 0, _mode = 0, _unused2 = '\000' <repeats 19 times>}
(gdb) n
14      FILE* fs2 = fdopen(fd, "r");
(gdb) p *fs1
$3 = {_flags = -72538999, _IO_read_ptr = 0x7ffffffffffdb0 "", _IO_read_end = 0x7ffffffffffdb0 "",
      _IO_read_base = 0x7ffffffffffdb0 "", _IO_write_base = 0x7ffffffffffdb0 "", _IO_write_ptr = 0x7ffffffffffdb0 "",
      _IO_write_end = 0x7ffffffffffdb0 "", _IO_buf_base = 0x7ffffffffffdb0 "", _IO_buf_end = 0x7ffffffffffdb0 "",
      _IO_backup_base = 0x0, _IO_save_end = 0x0, _markers = 0x0,
      _chain = 0x7ffff7f89440 <_IO_2_1_stderr_>, _fileno = 3, _flags2 = 0, _old_offset = 0, _cur_column = 0,
      _vtable_offset = 0 '\000', _shortbuf = "", _lock = 0x55555559380, _offset = -1, _codecvt = 0x0,
      _wide_data = 0x55555559390, _freeres_list = 0x0, _freeres_buf = 0x0, __pad5 = 0, _mode = 0,
      _unused2 = '\000' <repeats 19 times>}
(gdb) █
```

Рис. 1.4: Изменение полей структуры `FILE *fs1` при установке буфера

Следующий интересующий нас вызов — вызов функции стандартной библиотеки `fscanf()`. Первый раз он вызывается для `fs1` указателя на структуру `FILE`. При вызове `fscanf()` буфер `fs1` заполняется полностью, так как был выбран режим `_IOFBF` (полная буферизация). В структуре `struct file`, соответствующей дескриптору `fd`, поле `f_pos` увеличивается на 20 (из файла считалось 20 символов во время вызова `fscanf()`, чтобы заполнить буфер структуры `fs1`). Буфер `fs1` после 1 вызова `fscanf()` можно увидеть на рисунке 1.5. В нем видно, что буферизовался алфавит до буквы 't'.

```

21      flag1 = fscanf(fs1, "%c", &c);
(gdb) p *fs1
$4 = {_flags = -72538999, _IO_read_ptr = 0x7fffffffddb0 "", _IO_read_end = 0x7fffffffddb0 "",
      _IO_read_base = 0x7fffffffddb0 "", _IO_write_base = 0x7fffffffddb0 "", _IO_write_ptr = 0x7fffffffddb0 "",
      _IO_write_end = 0x7fffffffddb0 "", _IO_buf_base = 0x7fffffffddb0 "", _IO_buf_end = 0x7fffffffddc4 "",
      _IO_save_base = 0x0, _IO_backup_base = 0x0, _IO_save_end = 0x0, _markers = 0x0,
      _chain = 0x7ffff7f89440 <_IO_2_1_stderr->, _fileno = 3, _flags2 = 0, _old_offset = 0, _cur_column = 0,
      _vtable_offset = 0 '\000', _shortbuf = "", _lock = 0x55555559380, _offset = -1, _codecvt = 0x0,
      _wide_data = 0x55555559390, _freeres_list = 0x0, _freeres_buf = 0x0, __pad5 = 0, _mode = 0,
      _unused2 = '\000' <repeats 19 times>}
(gdb) n
22      if (flag1 == 1)
(gdb) p *fs1
$5 = {_flags = -72538999, _IO_read_ptr = 0x7fffffffddb1 "bcdefghijklmnopqrst", _IO_read_end = 0x7fffffffddc4 "",
      _IO_read_base = 0x7fffffffddb0 "bcdefghijklmnopqrst", _IO_write_base = 0x7fffffffddb0 "bcdefghijklmnopqrst",
      _IO_write_ptr = 0x7fffffffddb0 "bcdefghijklmnopqrst", _IO_write_end = 0x7fffffffddb0 "bcdefghijklmnopqrst",
      _IO_buf_base = 0x7fffffffddb0 "bcdefghijklmnopqrst", _IO_buf_end = 0x7fffffffddc4 "", _IO_save_base = 0x0,
      _IO_backup_base = 0x0, _IO_save_end = 0x0, _markers = 0x0, _chain = 0x7ffff7f89440 <_IO_2_1_stderr->,
      _fileno = 3, _flags2 = 0, _old_offset = 0, _cur_column = 0, _vtable_offset = 0 '\000', _shortbuf = "",
      _lock = 0x55555559380, _offset = -1, _codecvt = 0x0, _wide_data = 0x55555559390, _freeres_list = 0x0,
      _freeres_buf = 0x0, __pad5 = 0, _mode = -1, _unused2 = '\000' <repeats 19 times>}

```

Рис. 1.5: Изменение полей структуры `FILE *fs1` при первом вызове `fscanf()`

Когда `fscanf()` вызывается для `fs2` (учитывая, что структура `fs1` считала все символы до 't' и `fs2` имеет один файловый дескриптор со структурой `fs1`) буферизуются все символы после 't'. На рисунке 1.6 показано изменение полей структуры `FILE *fs2`.

Далее при следующих вызовах `fscanf()` для символов, символы будут браться из буфера до тех пор, пока он не станет пустым. Когда символы в буфере кончатся, структуры возвращают EOF, так как весь файл был прочитан (был дочитан структурой `fs2`).

```

24         flag2 = fscanf(fs2, "%c", &c);
(gdb) p *fs2
$1 = {_flags = -72538999, _IO_read_ptr = 0x7fffffffddd0 "", _IO_read_end = 0x7fffffffddd0 "",
      _IO_read_base = 0x7fffffffddd0 "", _IO_write_base = 0x7fffffffddd0 "", _IO_write_ptr = 0x7fffffffddd0 "",
      _IO_write_end = 0x7fffffffddd0 "", _IO_buf_base = 0x7fffffffddd0 "", _IO_buf_end = 0x7fffffffddde4 "\377\177",
      _IO_save_base = 0x0, _IO_backup_base = 0x0, _IO_save_end = 0x0, _markers = 0x0, _chain = 0x5555555592a0,
      _fileno = 3, _flags2 = 0, _old_offset = 0, _cur_column = 0, _vtable_offset = 0 '\000', _shortbuf = "",
      _lock = 0x555555559560, _offset = -1, _codecvt = 0x0, _wide_data = 0x555555559570, _freeres_list = 0x0,
      _freeres_buf = 0x0, __pad5 = 0, _mode = 0, _unused2 = '\000' <repeats 19 times>}
(gdb) n
25         if (flag2 == 1)
(gdb) p *fs2
$2 = {_flags = -72538999, _IO_read_ptr = 0x7fffffffddd1 "vwxyz", _IO_read_end = 0x7fffffffddd6 "",
      _IO_read_base = 0x7fffffffddd0 "uvwxyz", _IO_write_base = 0x7fffffffddd0 "uvwxyz",
      _IO_write_ptr = 0x7fffffffddd0 "uvwxyz", _IO_write_end = 0x7fffffffddd0 "uvwxyz",
      _IO_buf_base = 0x7fffffffddd0 "uvwxyz", _IO_buf_end = 0x7fffffffddde4 "\377\177", _IO_save_base = 0x0,
      _IO_backup_base = 0x0, _IO_save_end = 0x0, _markers = 0x0, _chain = 0x5555555592a0, _fileno = 3, _flags2 = 0,
      _old_offset = 0, _cur_column = 0, _vtable_offset = 0 '\000', _shortbuf = "", _lock = 0x555555559560,
      _offset = -1, _codecvt = 0x0, _wide_data = 0x555555559570, _freeres_list = 0x0, _freeres_buf = 0x0, __pad5 = 0,
      _mode = -1, _unused2 = '\000' <repeats 19 times>}

```

Рис. 1.6: Изменение полей структуры FILE \*fs2 при первом вызове fscanf()

## 2 Вторая программа

### 2.1 Код

#### 2.1.1 Один поток

```
1 #include <fcntl.h>
2 #include <unistd.h>
3
4 #define FILENAME "alphabet.txt"
5
6 int main() {
7     char c;
8     int fd1 = open(FILENAME, O_RDONLY);
9     int fd2 = open(FILENAME, O_RDONLY);
10
11     while (1) {
12         if (read(fd1, &c, 1) != 1)
13             break;
14         write(1, &c, 1);
15         if (read(fd2, &c, 1) != 1)
16             break;
17         write(1, &c, 1);
18     }
19     return 0;
20 }
```

Листинг 2.1: Вторая программа с одним потоком

#### 2.1.2 Два потока

```
1 #include <fcntl.h>
2 #include <unistd.h>
3 #include <pthread.h>
4 /*#include <stdio.h>*/
5
6 #define FILENAME "alphabet.txt"
7
8 void *run(void *arg) {
9     /*printf("\n=== Separate thread start ===\n");*/
10    int fd = *(int*)arg;
11    int flag = 1;
```

```

12     char c;
13     while (flag == 1) {
14         flag = read(fd, &c, 1);
15         if (flag == 1)
16             write(1, &c, 1);
17     }
18     /*printf("\n=== Separate thread end ===\n");*/
19     return NULL;
20 }
21
22 int main() {
23     /*setbuf(stdout, NULL);*/
24     int fd1 = open(FILENAME, O_RDONLY);
25     int fd2 = open(FILENAME, O_RDONLY);
26
27     pthread_t td;
28     pthread_create(&td, NULL, run, &fd2);
29     /*printf("\n=== Main thread start ===\n");*/
30
31     int flag = 1;
32     char c;
33     while (flag == 1) {
34         flag = read(fd1, &c, 1);
35         if (flag == 1)
36             write(1, &c, 1);
37     }
38     /*printf("\n=== Main thread end ===\n");*/
39     pthread_join(td, NULL);
40     return 0;
41 }

```

Листинг 2.2: Вторая программа с двумя потоками

## 2.2 Результат и анализ

### 2.2.1 Связь структур во 2 программе

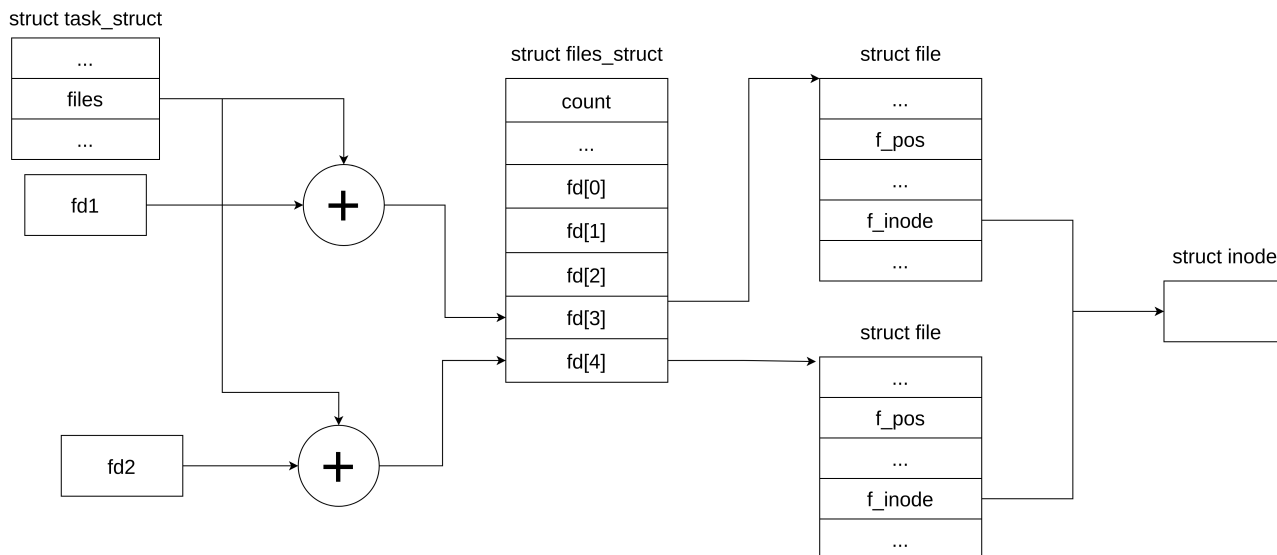


Рис. 2.1: Связь структур данных во 2 программе

### 2.2.2 С одним потоком

```
~ / ~/u/r/bmstu_os/sem_02/lab_05/src / ~ master !3 ?2 gcc p2.c
~ / ~/u/r/bmstu_os/sem_02/lab_05/src / ~ master !3 ?2 ./a.out
AAAbbccddeeffgghhiijjkkllmmnnooppqrrssttuuvvwxxyyzz
```

Рис. 2.2: Результат работы 2 программы с 1 потоком

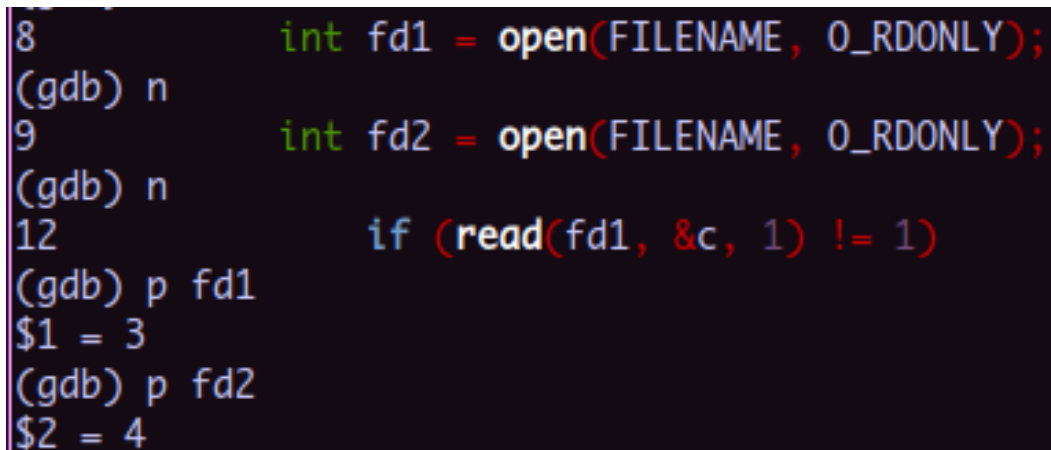
### 2.2.3 С двумя потоками

```
~ / ~/u/r/bmstu_os/sem_02/lab_05/src / ~ master !3 ?2 gcc p2mult.c -pthread
~ / ~/u/r/bmstu_os/sem_02/lab_05/src / ~ master !3 ?2 ./a.out
AbcdAebfcgdheifjgkhlminjokplqmrnsotpuqvrwsxytyuzvwxyz
```

Рис. 2.3: Результат работы 2 программы с 2 потоками

## 2.2.4 Анализ

Во второй программе работа с файлом происходит напрямую через файловые дескрипторы, но так как дескрипторы создаются дважды (см. рисунок 2.4, дескрипторы разные), то в программе имеются 2 различные `struct file`, имеющие одинаковые поля `struct inode *f_inode`. Так как структуры разные, то посимвольная печать просто выведет содержание файла дважды, причем в однопоточной реализации вывод будет вида 'AAbbcc...', а в случае с многопоточной реализацией вывод второго потока начнется чуть позже (так как на создание потока требуется время), поэтому содержимое файла будет полностью дважды, но выводы потоков перемешаются (что можно увидеть в результате работы для многопоточной реализации).



```
8      int fd1 = open(FILENAME, O_RDONLY);
(gdb) n
9      int fd2 = open(FILENAME, O_RDONLY);
(gdb) n
12     if (read(fd1, &c, 1) != 1)
(gdb) p fd1
$1 = 3
(gdb) p fd2
$2 = 4
```

Рис. 2.4: Разные дескрипторы в программе 2



## 3 Третья программа

### 3.1 Код

#### 3.1.1 Один поток

```
1 #include <fcntl.h>
2 #include <unistd.h>
3 #include <stdio.h>
4
5 #define FILENAME "out.txt"
6
7 int main() {
8     FILE *fd1 = fopen(FILENAME, "w");
9     FILE *fd2 = fopen(FILENAME, "w");
10    /*fseek(fd2, 1, SEEK_SET);*/
11    for (char c = 'a'; c <= 'z'; c++) {
12        if (c % 2) {
13            fprintf(fd1, "%c", c);
14        } else {
15            fprintf(fd2, "%c", c);
16        }
17    }
18    /*printf("from 'a': %d, from 'b': %d\n", fileno(fd1), fileno(fd2));*/
19    fclose(fd1);
20    fclose(fd2);
21    return 0;
22 }
```

Листинг 3.1: Третья программа с одним потоком

#### 3.1.2 Два потока

```
1 #include <fcntl.h>
2 #include <unistd.h>
3 #include <stdio.h>
4 #include <pthread.h>
5
6 #define FILENAME "out.txt"
7
8 void *run(void *arg) {
9     /*printf("\n=== Separate thread start ===\n");*/
```

```

10 FILE *fd2 = arg;
11 for (char c = 'b'; c <= 'z'; c += 2) {
12     fprintf(fd2, "%c", c);
13 }
14 fclose(fd2);
15 /*printf("\n=== Separate thread end ===\n");*/
16 return NULL;
17 }
18
19 int main() {
20     FILE *fd1 = fopen(FILENAME, "w");
21     FILE *fd2 = fopen(FILENAME, "w");
22
23     pthread_t td;
24     pthread_create(&td, NULL, run, fd2);
25
26     /*printf("\n=== Main thread start ===\n");*/
27     for (char c = 'a'; c <= 'z'; c += 2) {
28         fprintf(fd1, "%c", c);
29     }
30     /*sleep(1);*/
31     /*printf("\n=== Main thread end ===\n");*/
32
33     fclose(fd1);
34     pthread_join(td, NULL);
35     return 0;
36 }

```

Листинг 3.2: Третья программа с двумя потоками

## 3.2 Результат и анализ

### 3.2.1 Связь структур в 3 программе

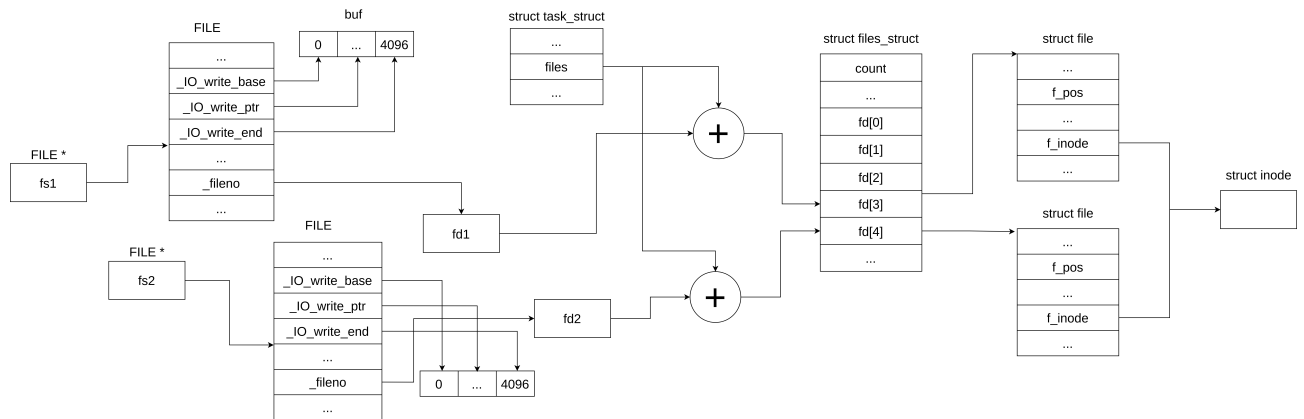


Рис. 3.1: Связь структур данных в 3 программе

### 3.2.2 С одним потоком

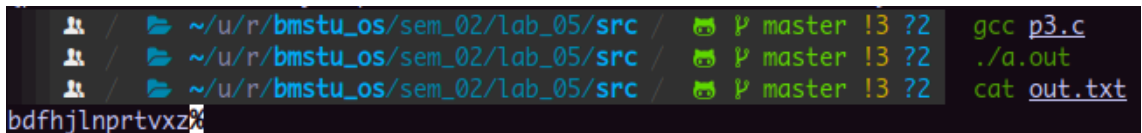


Рис. 3.2: Результат работы 3 программы с 1 потоком

### 3.2.3 С двумя потоками

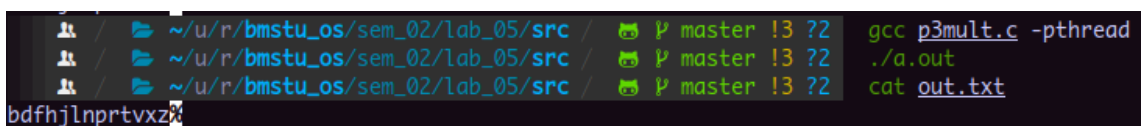


Рис. 3.3: Результат работы 3 программы с 2 потоками

### 3.2.4 Анализ

В 3 программе работа с файлом происходит с помощью функций стандартной библиотеки и структур `FILE *`. С помощью функции `fopen()` файл `alphabet.txt` открывается дважды на запись (`mode = 'w'`). При создании структуры не обладают никаким буфером (см. рисунок 3.4).

```
Breakpoint 1, main () at p3.c:8
8      FILE *fd1 = fopen(FILENAME, "w");
(gdb) n
9      FILE *fd2 = fopen(FILENAME, "w");
(gdb) p *fd1
$1 = {_flags = -72539004, _IO_read_ptr = 0x0, _IO_read_end = 0x0, _IO_read_base = 0x0, _IO_write_base = 0x0,
      _IO_write_ptr = 0x0, _IO_write_end = 0x0, _IO_buf_base = 0x0, _IO_buf_end = 0x0, _IO_save_base = 0x0,
      _IO_backup_base = 0x0, _IO_save_end = 0x0, _markers = 0x0, _chain = 0x7ffff7f89440 <_IO_2_1_stderr->,
      _fileno = 3, _flags2 = 0, _old_offset = 0, _cur_column = 0, _vtable_offset = 0 '\000', _shortbuf = "",
      _lock = 0x55555559380, _offset = -1, _codecvt = 0x0, _wide_data = 0x55555559390, _freeres_list = 0x0,
      _freeres_buf = 0x0, __pad5 = 0, _mode = 0, _unused2 = '\000' <repeats 19 times>}
(gdb) n
11     for (char c = 'a'; c <= 'z'; c++) {
(gdb) p *fd2
$2 = {_flags = -72539004, _IO_read_ptr = 0x0, _IO_read_end = 0x0, _IO_read_base = 0x0, _IO_write_base = 0x0,
      _IO_write_ptr = 0x0, _IO_write_end = 0x0, _IO_buf_base = 0x0, _IO_buf_end = 0x0, _IO_save_base = 0x0,
      _IO_backup_base = 0x0, _IO_save_end = 0x0, _markers = 0x0, _chain = 0x555555592a0, _fileno = 4, _flags2 = 0,
      _old_offset = 0, _cur_column = 0, _vtable_offset = 0 '\000', _shortbuf = "", _lock = 0x55555559560,
      _offset = -1, _codecvt = 0x0, _wide_data = 0x55555559570, _freeres_list = 0x0, _freeres_buf = 0x0, __pad5 = 0,
      _mode = 0, _unused2 = '\000' <repeats 19 times>}
(gdb)
```

Рис. 3.4: Изначальное состояние структур `FILE *fd1`, `*fd2`

Буфер создается при первой операции записи, размер буфера = 4096 байт (4 Кб, размер 1 страницы памяти). Изменение полей структуры при первой записи можно увидеть на рисунке 3.5.

```
13      fprintf(fd1, "%c", c);
(gdb) p *fd1
$3 = {_flags = -72539004, _IO_read_ptr = 0x0, _IO_read_end = 0x0, _IO_read_base = 0x0, _IO_write_base = 0x0,
      _IO_write_ptr = 0x0, _IO_write_end = 0x0, _IO_buf_base = 0x0, _IO_buf_end = 0x0, _IO_save_base = 0x0,
      _IO_backup_base = 0x0, _IO_save_end = 0x0, _markers = 0x0, _chain = 0x7ffff7f89440 <_IO_2_1_stderr->,
      _fileno = 3, _flags2 = 0, _old_offset = 0, _cur_column = 0, _vtable_offset = 0 '\000', _shortbuf = "",
      _lock = 0x55555559380, _offset = -1, _codecvt = 0x0, _wide_data = 0x55555559390, _freeres_list = 0x0,
      _freeres_buf = 0x0, __pad5 = 0, _mode = 0, _unused2 = '\000' <repeats 19 times>}
(gdb) n
11     for (char c = 'a'; c <= 'z'; c++) {
(gdb) p *fd1
$4 = {_flags = -72536956, _IO_read_ptr = 0x55555559660 "a", _IO_read_end = 0x55555559660 "a",
      _IO_read_base = 0x55555559660 "a", _IO_write_base = 0x55555559660 "a", _IO_write_ptr = 0x55555559661 "",
      _IO_write_end = 0x55555559660 "", _IO_buf_base = 0x55555559660 "a", _IO_buf_end = 0x55555559660 "",
      _IO_save_base = 0x0, _IO_backup_base = 0x0, _IO_save_end = 0x0, _markers = 0x0,
      _chain = 0x7ffff7f89440 <_IO_2_1_stderr->, _fileno = 3, _flags2 = 0, _old_offset = 0, _cur_column = 0,
      _vtable_offset = 0 '\000', _shortbuf = "", _lock = 0x55555559380, _offset = -1, _codecvt = 0x0,
      _wide_data = 0x55555559390, _freeres_list = 0x0, _freeres_buf = 0x0, __pad5 = 0, _mode = -1,
      _unused2 = '\000' <repeats 19 times>}
(gdb) p fd1->_IO_write_end - fd1->_IO_write_base
$5 = 4096
```

Рис. 3.5: Изменение полей структуры `fd1` при первой записи

В конце работы программы делается вызов функции стандартной библиотеки `fclose()`. Во время этого вызова в нашей программе содержимое

буфера переносится в файл (в общем случае это может происходить по одной из 3 причин: буфер полон, вызван `fflush()`, вызван `fclose()`). Так как `fclose(fd2)` вызывается позже, содержимое файла переписывается содержимым буфера структуры `fd2`. На рисунке 3.6 можно посмотреть, как меняется содержимое файла, на рисунках 3.7 и 3.8 можно посмотреть, как меняются поля структур `fd1`, `fd2` при вызове `fclose()`.

```

19      fclose(fd1);
(gdb) !cat out.txt
(gdb) n
20      fclose(fd2);
(gdb) !cat out.txt
acegikmoqsuwy(gdb) n
21      return 0;
(gdb) !cat out.txt
bdfhjlnprtvxz(gdb)

```

Рис. 3.6: Изменение содержимого файла при вызовах `fclose()`

```

19      fclose(fd1);
(gdb) p *fd1
$1 = {_flags = -72536956, _IO_read_ptr = 0x555555559660 "acegikmoqsuwy",
      _IO_read_end = 0x555555559660 "acegikmoqsuwy", _IO_read_base = 0x555555559660 "acegikmoqsuwy",
      _IO_write_base = 0x555555559660 "acegikmoqsuwy", _IO_write_ptr = 0x55555555966d "",
      _IO_write_end = 0x55555555a660 "", _IO_buf_base = 0x555555559660 "acegikmoqsuwy",
      _IO_buf_end = 0x55555555a660 "", _IO_save_base = 0x0, _IO_backup_base = 0x0, _IO_save_end = 0x0, _markers = 0x0,
      _chain = 0x7ffff7f89440 <_IO_2_1_stderr_>, _fileno = 3, _flags2 = 0, _old_offset = 0, _cur_column = 0,
      _vtable_offset = 0 '\000', _shortbuf = "", _lock = 0x555555559380, _offset = -1, _codecvt = 0x0,
      _wide_data = 0x555555559390, _freeres_list = 0x0, _freeres_buf = 0x0, __pad5 = 0, _mode = -1,
      _unused2 = '\000' <repeats 19 times>}
(gdb) n
20      fclose(fd2);
(gdb) p *fd1
$2 = {_flags = 1431655769, _IO_read_ptr = 0x555555559010 "", _IO_read_end = 0x0, _IO_read_base = 0x0,
      _IO_write_base = 0x0, _IO_write_ptr = 0x0, _IO_write_end = 0x0, _IO_buf_base = 0x0, _IO_buf_end = 0x0,
      _IO_save_base = 0x0, _IO_backup_base = 0x0, _IO_save_end = 0x0, _markers = 0x0,
      _chain = 0x7ffff7f89440 <_IO_2_1_stderr_>, _fileno = -1, _flags2 = 0, _old_offset = 0, _cur_column = 0,
      _vtable_offset = 0 '\000', _shortbuf = "", _lock = 0x555555559380, _offset = -1, _codecvt = 0x0,
      _wide_data = 0x555555559390, _freeres_list = 0x0, _freeres_buf = 0x0, __pad5 = 0, _mode = -1,
      _unused2 = '\000' <repeats 19 times>}

```

Рис. 3.7: Изменение полей структуры `fd1` вызове `fclose(fd1)`

```

20      fclose(fd2);
(gdb) p *fd2
$3 = {_flags = -72536956, _IO_read_ptr = 0x5555555a670 "bdfhjlnprtvxz",
      _IO_read_end = 0x55555555a670 "bdfhjlnprtvxz", _IO_read_base = 0x55555555a670 "bdfhjlnprtvxz",
      _IO_write_base = 0x55555555a670 "bdfhjlnprtvxz", _IO_write_ptr = 0x55555555a67d "",
      _IO_write_end = 0x55555555b670 "", _IO_buf_base = 0x55555555a670 "bdfhjlnprtvxz",
      _IO_buf_end = 0x55555555b670 "", _IO_save_base = 0x0, _IO_backup_base = 0x0, _IO_save_end = 0x0, _markers = 0x0,
      _chain = 0x7ffff7f89440 <_IO_2_1_stderr->, _fileno = 4, _flags2 = 0, _old_offset = 0, _cur_column = 0,
      _vtable_offset = 0 '\000', _shortbuf = "", _lock = 0x555555559560, _offset = -1, _codecvt = 0x0,
      _wide_data = 0x555555559570, _freeres_list = 0x0, _freeres_buf = 0x0, __pad5 = 0, _mode = -1,
      _unused2 = '\000' <repeats 19 times>}}
(gdb) n
21      return 0;
(gdb) p *fd2
$4 = {_flags = 51193, _IO_read_ptr = 0x555555559010 "", _IO_read_end = 0x0, _IO_read_base = 0x0,
      _IO_write_base = 0x0, _IO_write_ptr = 0x0, _IO_write_end = 0x0, _IO_buf_base = 0x0, _IO_buf_end = 0x0,
      _IO_save_base = 0x0, _IO_backup_base = 0x0, _IO_save_end = 0x0, _markers = 0x0,
      _chain = 0x7ffff7f89440 <_IO_2_1_stderr->, _fileno = -1, _flags2 = 0, _old_offset = 0, _cur_column = 0,
      _vtable_offset = 0 '\000', _shortbuf = "", _lock = 0x555555559560, _offset = -1, _codecvt = 0x0,
      _wide_data = 0x555555559570, _freeres_list = 0x0, _freeres_buf = 0x0, __pad5 = 0, _mode = -1,
      _unused2 = '\000' <repeats 19 times>}}

```

Рис. 3.8: Изменение полей структуры fd2 вызове fclose(fd2)

Замечания:

- Если не вызвать `fclose`, на моей системе содержимое буфера дескриптора с меньшим значением будет записано в файл.
- Если один из дескрипторов сдвинуть (через `fseek`), то перезапишутся только пересекающие диапазоны.