



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 Богаченко А.Е.

Группа ИУ7-65Б

Оценка (баллы) \_\_\_\_\_

Преподаватели Рязанова Н. Ю.

# 1. Структура \_IO\_FILE

Листинг 1 – Структура \_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    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  };
```

## 2. Первая программа

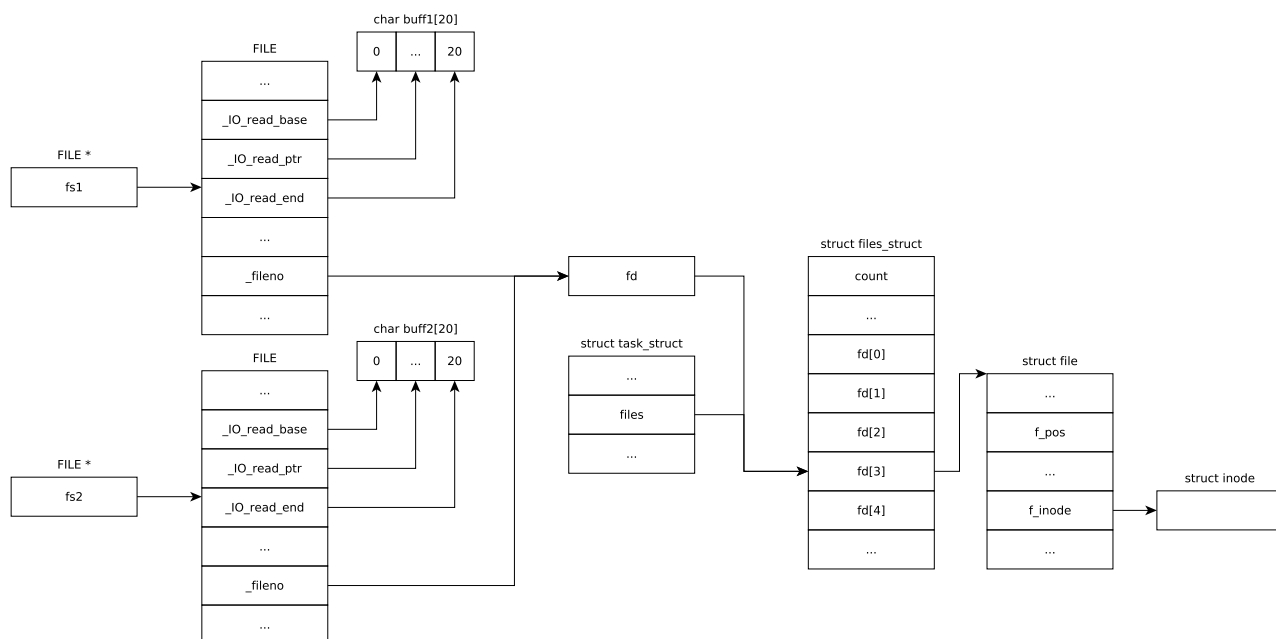
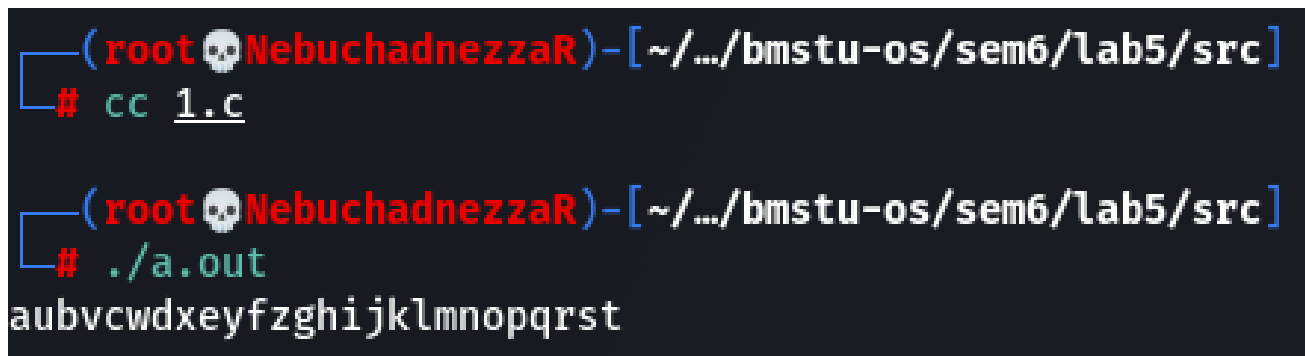


Рисунок 1 – Используемые структуры

- Функция `open()` создает новый файловый дескриптор `fd` файла (открытого только на чтение) `"alphabet.txt"` запись в системной таблице открытых файлов. Эта запись регистрирует смещение в файле и флаги состояния файла
- Функция `fdopen()` создаёт 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`) буффер `buff2` будет заполнен оставшимися 6 символами (начиная с `f_pos`)
- В цикле поочерёдно выводятся символы из `buff1` и `buff2`

## Листинг 2 – Исходный код первой программы

```
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) fprintf(stdout, "%c", c);
23
24        flag2 = fscanf(fs2, "%c", &c);
25        if (flag2 == 1) fprintf(stdout, "%c", c);
26    }
27
28    return 0;
29 }
```



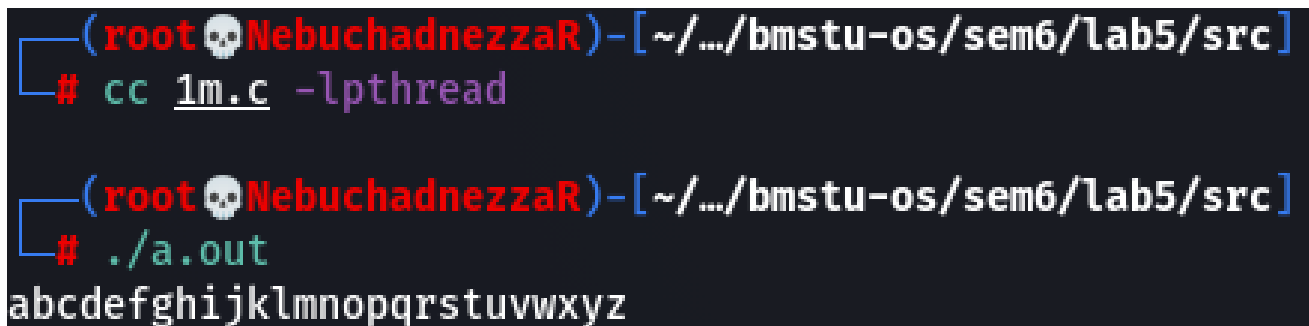
```
(root NebuchadnezzaR)-[~/.../bmstu-os/sem6/lab5/src]
# cc 1.c

(root NebuchadnezzaR)-[~/.../bmstu-os/sem6/lab5/src]
# ./a.out
aubvcwdxeyfzghijklmnopqrst
```

Рисунок 2 – Результат работы первой программы

### Листинг 3 – Исходный код первой программы (многопоточная)

```
1 #include <fcntl.h>
2 #include <pthread.h>
3 #include <stdio.h>
4 #define BUF_SIZE 20
5 #define FILENAME "alphabet.txt"
6
7 void *run(void *args) {
8     int *fd = (int *)args;
9     FILE *fs2 = fdopen(*fd, "r");
10    char buff2[BUF_SIZE];
11    setvbuf(fs2, buff2, _IOFBF, BUF_SIZE);
12    int flag = 1;
13    char c;
14    while ((flag = fscanf(fs2, "%c", &c)) == 1) {
15        fprintf(stdout, "%c", c);
16    }
17    return NULL;
18 }
19
20 int main() {
21    pthread_t td;
22    int fd = open(FILENAME, O_RDONLY);
23
24    FILE *fs1 = fdopen(fd, "r");
25    char buff1[BUF_SIZE];
26    setvbuf(fs1, buff1, _IOFBF, BUF_SIZE);
27    pthread_create (&td, NULL, run, &fd);
28    int flag = 1;
29    char c;
30    while ((flag = fscanf(fs1, "%c", &c)) == 1) {
31        fprintf(stdout, "%c", c);
32    }
33    pthread_join(td, NULL);
34    return 0;
35 }
```



```
(root NebuchadnezzaR) - [~/.../bmstu-os/sem6/lab5/src]
# cc 1m.c -lpthread

(root NebuchadnezzaR) - [~/.../bmstu-os/sem6/lab5/src]
# ./a.out
abcdefghijklmnopqrstuvwxyz
```

Рисунок 3 – Результат работы первой программы (многопоточная)

### 3. Вторая программа

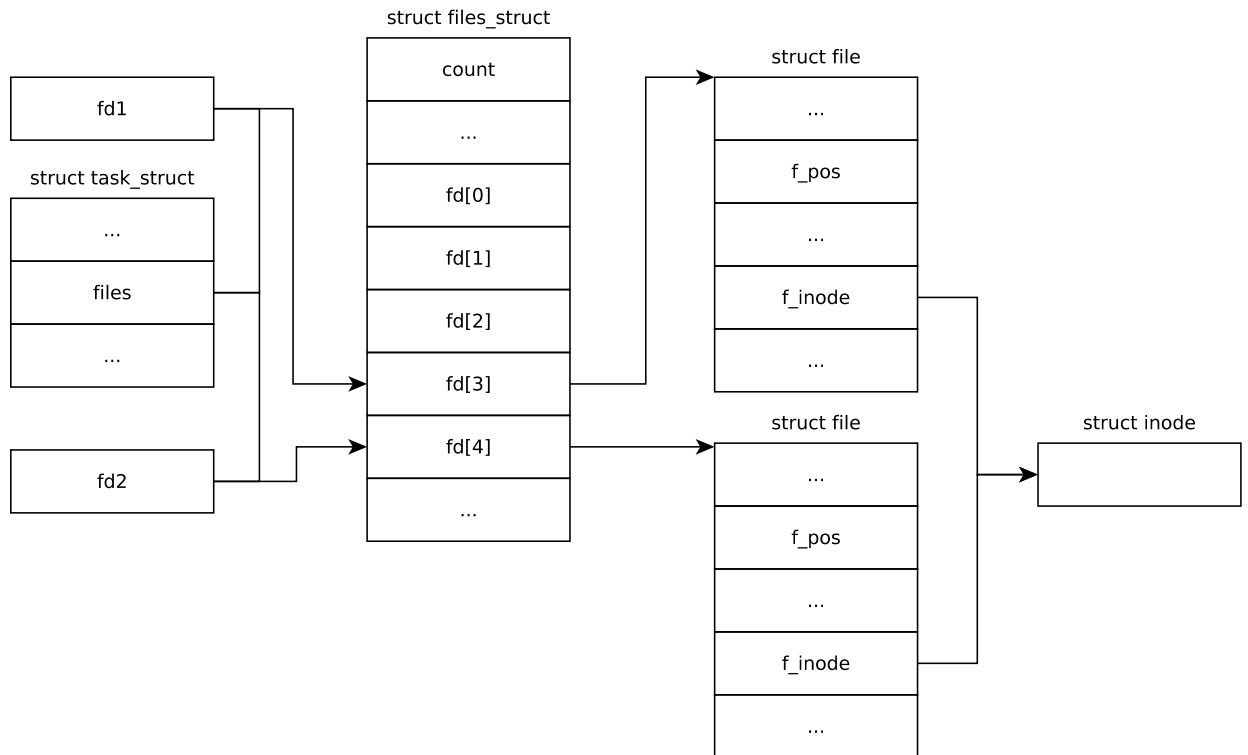
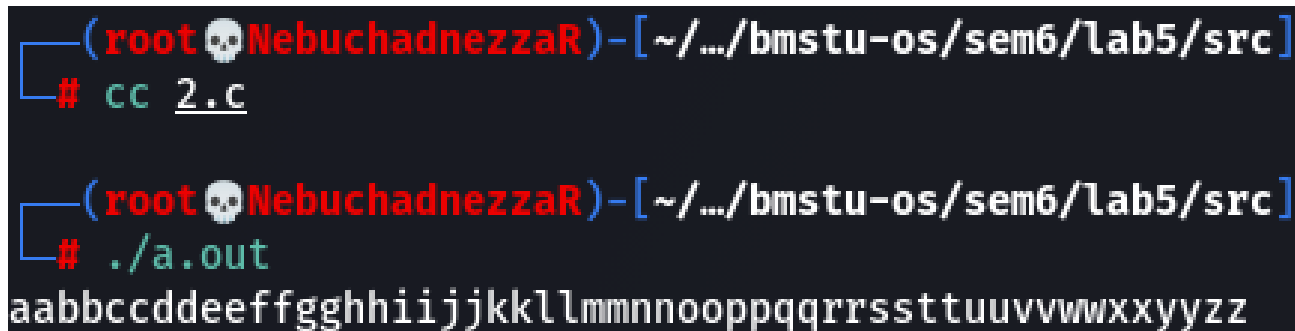


Рисунок 4 – Используемые структуры

- Функция `open()` создаёт 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 «`aabbcc...`» (в случае однопоточной реализации);
- В случае многопоточной реализации, вывод второго потока начнётся позже (нужно время, для создание этого потока) и символы перемешаются, что видно на рисунке 6

Листинг 4 – Исходный код второй программы

```
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) break;
13         write(1, &c, 1);
14
15         if (read(fd2, &c, 1) != 1) break;
16         write(1, &c, 1);
17     }
18
19     return 0;
20 }
```



```
(root👤NebuchadnezzaR)-[~/.../bmstu-os/sem6/lab5/src]
# cc 2.c

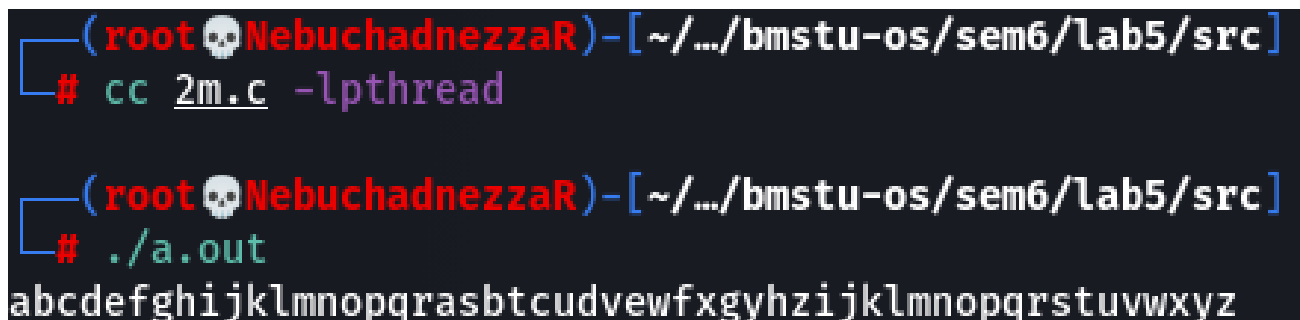
(root👤NebuchadnezzaR)-[~/.../bmstu-os/sem6/lab5/src]
# ./a.out
aabbccddeeffgghhiijjkkllmmnnnooppqrrssttuuvvwxxyyzz
```

Рисунок 5 – Результат работы второй программы



Листинг 5 – Исходный код второй программы (многопоточная)

```
1 #include <fcntl.h>
2 #include <pthread.h>
3 #include <unistd.h>
4
5 #define FILENAME "alphabet.txt"
6
7 void *run(void *args) {
8     int fd = open(FILENAME, O_RDONLY);
9     int flag = 1;
10    char c;
11    while ((flag = read(fd, &c, 1)) == 1) {
12        write(1, &c, 1);
13    }
14    return NULL;
15 }
16
17 int main() {
18     int fd1 = open(FILENAME, O_RDONLY);
19     pthread_t td;
20     pthread_create(&td, NULL, run, NULL);
21
22     int flag = 1;
23     char c;
24     while ((flag = read(fd1, &c, 1)) == 1) {
25         write(1, &c, 1);
26     }
27
28     pthread_join(td, NULL);
29
30     return 0;
31 }
```



```
(root👤NebuchadnezzaR)-[~/.../bmstu-os/sem6/lab5/src]
# cc 2m.c -lpthread

(root👤NebuchadnezzaR)-[~/.../bmstu-os/sem6/lab5/src]
# ./a.out
abcdefghijklmnopqrstuvwxyz
```

Рисунок 6 – Результат работы второй программы (многопоточная)

### 3. Третья программа

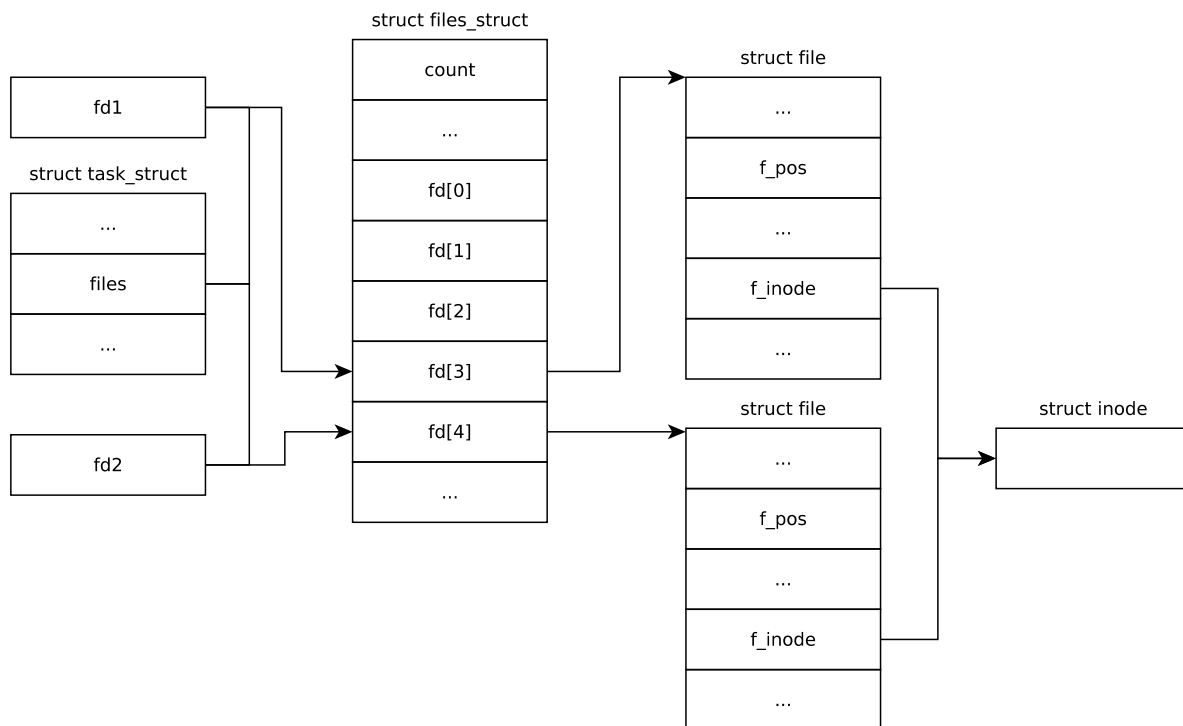
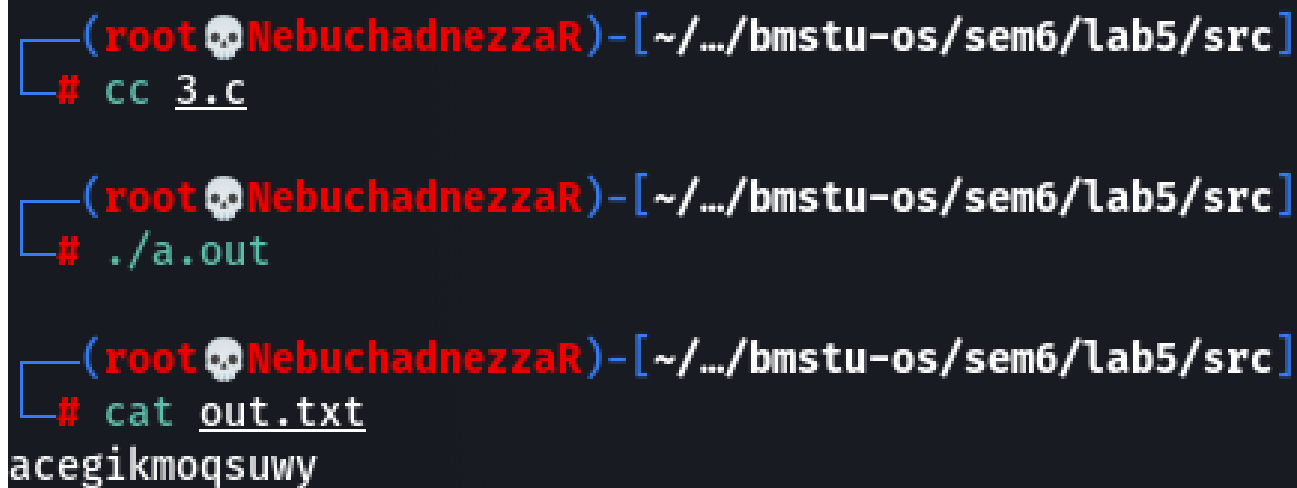


Рисунок 7 – Используемые структуры

- Файл открывается на запись два раза, с помощью функции `fopen()`
- Функция `fprintf()` предоставляет буферизованный вывод - буфер создаётся без нашего вмешательства
- Изначально информация пишется в буфер, а из буфера в файл если произошло одно из событий:
  1. буфер полон
  2. вызвана функция `fclose()`
  3. 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 производиться с самого начала

## Листинг 6 – Исходный код Третьей программы

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



```
(root 🐼 NebuchadnezzaR) - [~/.../bmstu-os/sem6/lab5/src]
# cc 3.c

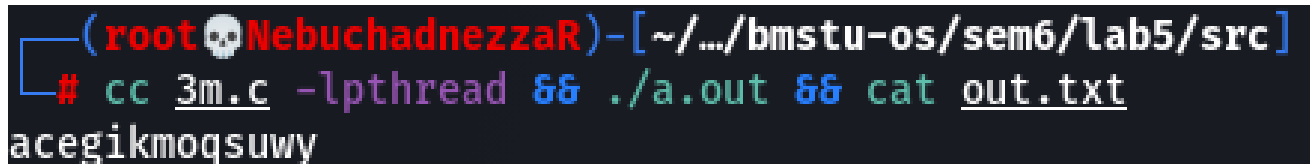
(root 🐼 NebuchadnezzaR) - [~/.../bmstu-os/sem6/lab5/src]
# ./a.out

(root 🐼 NebuchadnezzaR) - [~/.../bmstu-os/sem6/lab5/src]
# cat out.txt
acegikmoqsuwy
```

Рисунок 8 – Результат работы третьей программы

### Листинг 7 – Исходный код третьей программы (многопоточная)

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



```
(root NebuchadnezzaR)-[~/.../bmstu-os/sem6/lab5/src]
# cc 3m.c -lpthread && ./a.out && cat out.txt
acegikmoqsuwy
```

Рисунок 9 – Результат работы третьей программы (многопоточная)