**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, искусственный интелект и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

# Отчет по лабораторной работе № 8

# по курсу «Операционные системы»

**Тема** Буферизованный и не буферизованный ввод-вывод.

**Студент** Авсюнин А. А.

**Группа** ИУ7-66Б

**Оценка (баллы)** _____

**Преподаватель** Рязанова Н.Ю.

Москва — 2023 г.

## Используемые структуры

Листинг 1 – Структура _IO_FILE

```
1 struct _IO_FILE {
2     int _flags;           /* High-order word is _IO_MAGIC; rest is flags.
     */
3     #define _IO_file_flags _flags
4     /* The following pointers correspond to the C++ streambuf protocol. */
5     /* Note:  Tk uses the _IO_read_ptr and _IO_read_end fields directly.
     */
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    /* The following fields are used to support backing up and undo. */
15    char *_IO_save_base; /* Pointer to start of non-current get area. */
16    char *_IO_backup_base;  /* Pointer to first valid character of backup
     area */
17    char *_IO_save_end; /* Pointer to end of non-current get area. */
18    struct _IO_marker *_markers;
19    struct _IO_FILE *_chain;
20    int _fileno;
21    #if 0
22    int _blksize;
23    #else
24    int _flags2;
25    #endif
26    _IO_off_t _old_offset; /* This used to be _offset but it's too small.
      */
27    #define __HAVE_COLUMN /* temporary */
28    /* 1+column number of pbase(); 0 is unknown. */
29    unsigned short _cur_column;
30    signed char _vtable_offset;
31    char _shortbuf[1];
32    /*  char* _save_gptr;  char* _save_egptr; */
33    _IO_lock_t *_lock;
34    #ifdef _IO_USE_OLD_IO_FILE
35 };
```

Листинг 2 – Структура filename

```
struct stat {
    dev_t     st_dev;         /* device */
    ino_t     st_ino;         /* inode */
    mode_t    st_mode;        /* access rules */
    nlink_t   st_nlink;       /* links */
    uid_t     st_uid;         /* UID*/
    gid_t     st_gid;         /* GID*/
    dev_t     st_rdev;        /* ID device (special file) */
    off_t     st_size;        /* byte size */
    blksize_t st_blksize;     /* size of system IO block */
    blkcnt_t  st_blocks;      /* count of allocated blocks per 512 bytes
    */
    struct timespec st_atim;  /* last launch */
    struct timespec st_mtim;  /* last update */
    struct timespec st_ctim;  /* last change */
    #define st_atime st_atim.tv_sec      /* backward compatibility */
    #define st_mtime st_mtim.tv_sec
    #define st_ctime st_ctim.tv_sec
};
```

# Программа 1

В данной программе создаётся 1 дескриптор файла и на его основе создаются 2 объекта структуры struct _IO_FILE. Устанавливается буфер размером 20 байт и происходит поочерёдное обращение к каждой структуре для чтения одного символа из файла и вывода его на экран. Чтение продолжается пока есть возможность читать, обращаясь хотя бы к одной из структур.

**Однопоточный вариант**

Листинг 3 – Однопоточный вариант

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

int main()
{
    int fd = open("alphabet.txt",O_RDONLY);
    FILE *fs1 = fdopen(fd,"r");
    char buff1[20];
    setvbuf(fs1,buff1,_IOFBF,20); // full buffering
    FILE *fs2 = fdopen(fd,"r");
    char buff2[20];
    setvbuf(fs2,buff2,_IOFBF,20); // full buffering\\
```

Листинг 4 – Однопоточный вариант

```
1     int flag1 = 1, flag2 = 1;
2     while(flag1 == 1 || flag2 == 1)
3     {
4         char c;
5         flag1 = fscanf(fs1,"%c",&c);
6         if (flag1 == 1)
7         {
8             fprintf(stdout,"%c",c);
9         }
10        flag2 = fscanf(fs2,"%c",&c);
11        if (flag2 == 1)
12        {
13            fprintf(stdout,"%c",c);
14        }
15    }
16    return 0;
17 }
```

Результат работы однопоточного варианта представлен на рисунке 1.

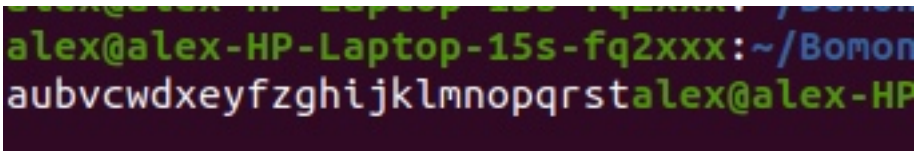


Рисунок 1 – Результат работы однопоточной программы

### Многопоточный вариант

Листинг 5 – Многопоточный вариант

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <pthread.h>
4
5 void *thread(void *args)
6 {
7     int fd = *(int *) args;
8     FILE *fs = fdopen(fd,"r");
9     char buff[20];
10    setvbuf(fs,buff,_IOFBF,20); // full buffering
11    int flag = 1;
12    while(flag == 1)
13    {
14        char c;
15        flag = fscanf(fs,"%c",&c);
```

Листинг 6 – Многопоточный вариант

```
1           if (flag == 1)
2           {
3               fprintf(stdout,"%c",c);
4           }
5       }
6 }
7
8 int main()
9 {
10      int fd = open("alphabet.txt",O_RDONLY);
11      pthread_t th1;
12      pthread_t th2;
13      pthread_create(&th1, NULL, thread, &fd);
14      pthread_create(&th2, NULL, thread, &fd);
15      pthread_join(th1, NULL);
16      pthread_join(th2, NULL);
17      return 0;
18 }
```

Результат работы многопоточного варианта представлен на рисунке 2.

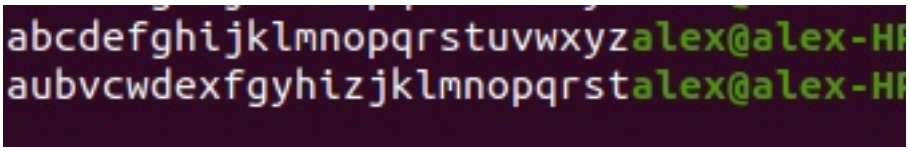


Рисунок 2 – Результат работы многопоточной программы

# Анализ результата

При первом вызове функции fscanf буфер затронутого экземпляра структуры _IO_FILE полностью заполняется 20 символами (от a до t), а значит файловый указатель перемещается на 20 символов вправо. Второй вызов fscanf с другим экземпляром структуры _IO_FILE также заполняет его буфер максимальным оставшимся количеством символов (от t до z) и перемещает файловый указатель в конец файла. Дальнейшие вызовы никак не меняют состояние буферов, так как данные из файла уже считаны. При вызове fprintf буквы из каждого буфера поочерёдно выводятся на экран. Таким образом, полученный результат связан исключительно с буферизацией ввода.

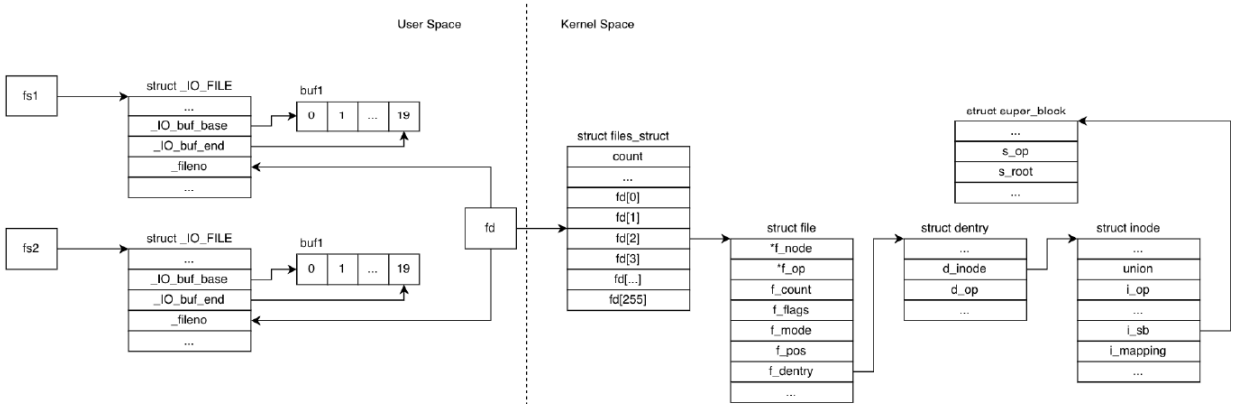


Рисунок 3 – Схема связи структур

# Программа 2

В данной программе создаётся два дескриптора одного файла. Далее происходит поочерёдное обращение к каждому дескриптору для чтения одного символа из файла и вывода одного символа на экран. Чтение продолжается пока есть возможность читать, обращаясь к каждому дескриптору.

**Однопоточный вариант**

Листинг 7 – Однопоточный вариант

```
#include <unistd.h>
#include <fcntl.h>

int main()
{
    char c;
    int fd1 = open("alphabet.txt",O_RDONLY);
    int fd2 = open("alphabet.txt",O_RDONLY);
    int flag = 1;
    while(flag == 1)
    {
        if ((flag = read(fd1,&c,1)) == 1)
            write(1,&c,1);
        if (flag == 1 && (flag = read(fd2,&c,1)) == 1)
            write(1,&c,1);
    }
    close(fd1);
    close(fd2);
    return 0;
}
```

Результат работы однопоточного варианта представлен на рисунке 4.

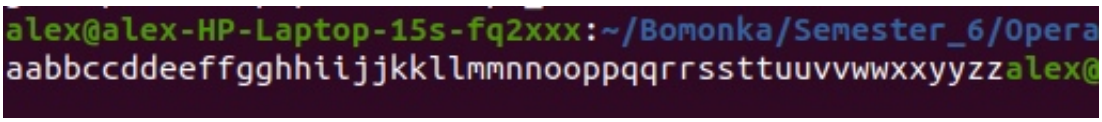


Рисунок 4 – Результат работы однопоточной программы

### Многопоточный вариант

Листинг 8 – Многопоточный вариант

```
1 #include <unistd.h>
2 #include <fcntl.h>
3 #include <pthread.h>
4
5 void *thread(void *args)
6 {
7     char c;
8     int fd = open("alphabet.txt",O_RDONLY);
9     int flag = 1;
10    while (flag == 1)
11    {
12        if ((flag = read(fd, &c, 1)) == 1)
13        write(1, &c, 1);
14    }
15    close(fd);
16    return NULL;
17 }
18
19 int main()
20 {
21    pthread_t th1;
22    pthread_t th2;
23    pthread_create(&th1, NULL, thread, NULL);
24    pthread_create(&th2, NULL, thread, NULL);
25    pthread_join(th1, NULL);
26    pthread_join(th2, NULL);
27    return 0;
28 }
```

Результат работы многопоточного варианта представлен на рисунке 5.

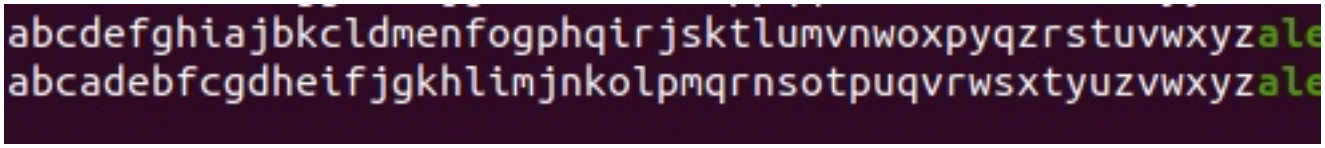


Рисунок 5 – Результат работы многопоточной программы

# Анализ результата

Чтение символа из файла для каждого дескриптора происходит независимо от другого, для каждого дескриптора существует файловый указатель, поэтому в результате получается удвоение каждого символа. Таким образом файл читается полностью 2 раза.

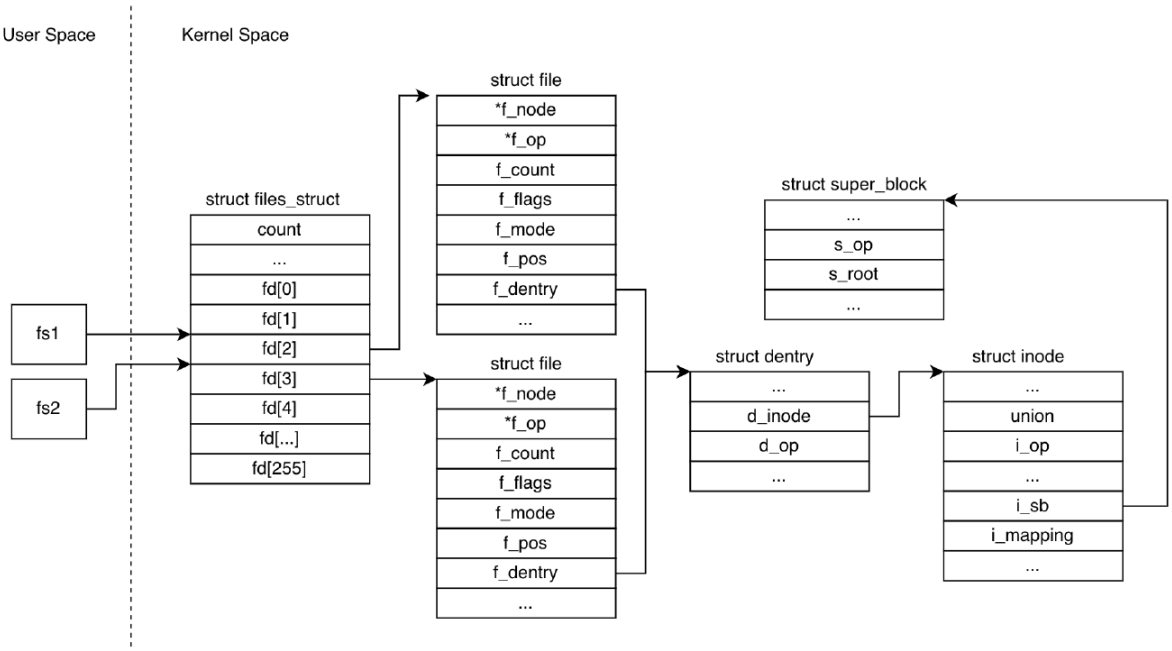


Рисунок 6 – Схема связи структур

# Программа 3

В данной программе открывается один и тот же файл два раза с использованием библиотечной функции fopen(). Для этого объявляются два файловых дескриптора типа FILE. В цикле записываются в файл буквы латинского алфавита, поочередно передавая функции fprintf() то первый дескриптор, то – второй.

### Однопоточный вариант

Листинг 9 – Однопоточный вариант

```
1 #include <stdio.h>
2 #include <sys/stat.h>
3 #include <unistd.h>
4
5 int main()
6 {
7     FILE *fs1 = fopen("new_alphabet.txt", "a");
8     FILE *fs2 = fopen("new_alphabet.txt", "a");
9     char c = 'a';
10    int flag = 0;
11    while (c <= 'z')
12        (flag = !flag) ? fprintf(fs1, "%c", c++) : fprintf(fs2, "%c", c++)
      ;
13    struct stat filestat;
14    fstat(fs1->_fileno, &filestat);
15    printf("inode=%lu, size=%ld\n", filestat.st_ino, filestat.st_size);
16    fclose(fs1);
17
18    fstat(fs2->_fileno, &filestat);
19    printf("inode=%lu, size=%ld\n", filestat.st_ino, filestat.st_size);
20    fclose(fs2);
21
22    stat("new_alphabet.txt", &filestat);
23    printf("inode=%lu, size=%ld\n", filestat.st_ino, filestat.st_size);
24    return 0;
25 }
```

Результат работы однопоточного варианта программы представлен на рисунках 7, 8.

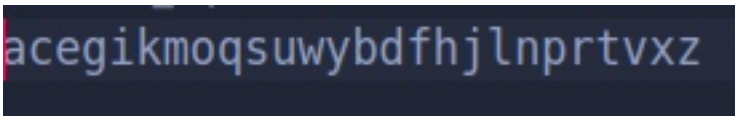


Рисунок 7 – Результат работы однопоточной программы

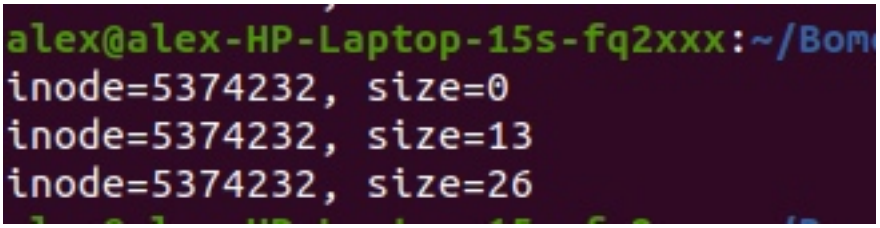


Рисунок 8 – Результат работы однопоточной программы

### Многопоточный вариант

Листинг 10 – Многопоточный вариант

```
 1 #include <stdio.h>
 2 #include <pthread.h>
 3 #include <sys/stat.h>
 4 #include <unistd.h>
 5
 6 void *thread(void *args)
 7 {
 8     int shift = *(int *) args;
 9     FILE *fs = fopen("new_alphabet.txt", "a");
10     char c = 'a' + shift;
11     while (c <= 'z')
12     {
13         fprintf(fs, "%c", c);
14         c += 2;
15     }
16     struct stat filestat;
17     fstat(fs->_fileno, &filestat);
18     printf("shift=%d, inode=%lu, size=%ld\n", shift, filestat.st_ino,
      filestat.st_size);
19     fclose(fs);
20     stat("new_alphabet.txt", &filestat);
21     printf("shift=%d, inode=%lu, size=%ld\n", shift, filestat.st_ino,
      filestat.st_size);
22     return NULL;
23 }
24
25 int main()
26 {
27     pthread_t th1;
28     pthread_t th2;
29     int shift1 = 0;
30     int shift2 = 1;
31     pthread_create(&th1, NULL, thread, &shift1);
32     pthread_create(&th2, NULL, thread, &shift2);
33     pthread_join(th1, NULL);
34     pthread_join(th2, NULL);
35     return 0;
36 }
```

Результат работы многопоточного варианта программы представлен на рисунках 9, 10.

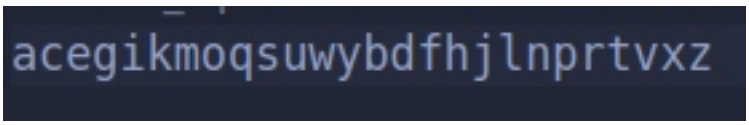


Рисунок 9 – Результат работы многопоточной программы

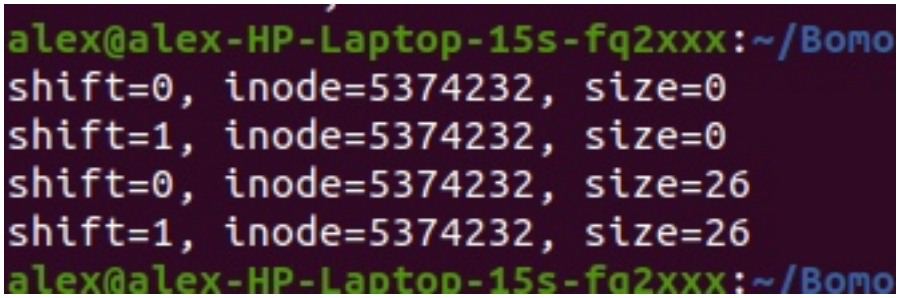


Рисунок 10 – Результат работы многопоточной программы

# Анализ результата

Вызов функции fprintf с экземпляром структуры _IO_FILE приводит к тому, что соответствующий этому экземпляру буфер начинает заполняться, а данные на экран не выводятся. Так происходит, пока буфер не будет заполнен, не будет вызвана функция fflush для принудительного сброса содержимого буфера или не файл не будет закрыт. В данном случае печать происходит при закрытии файла, это видно из информации приведённой в структуре stat. Размер файла до закрытия равен 0, а после 13.

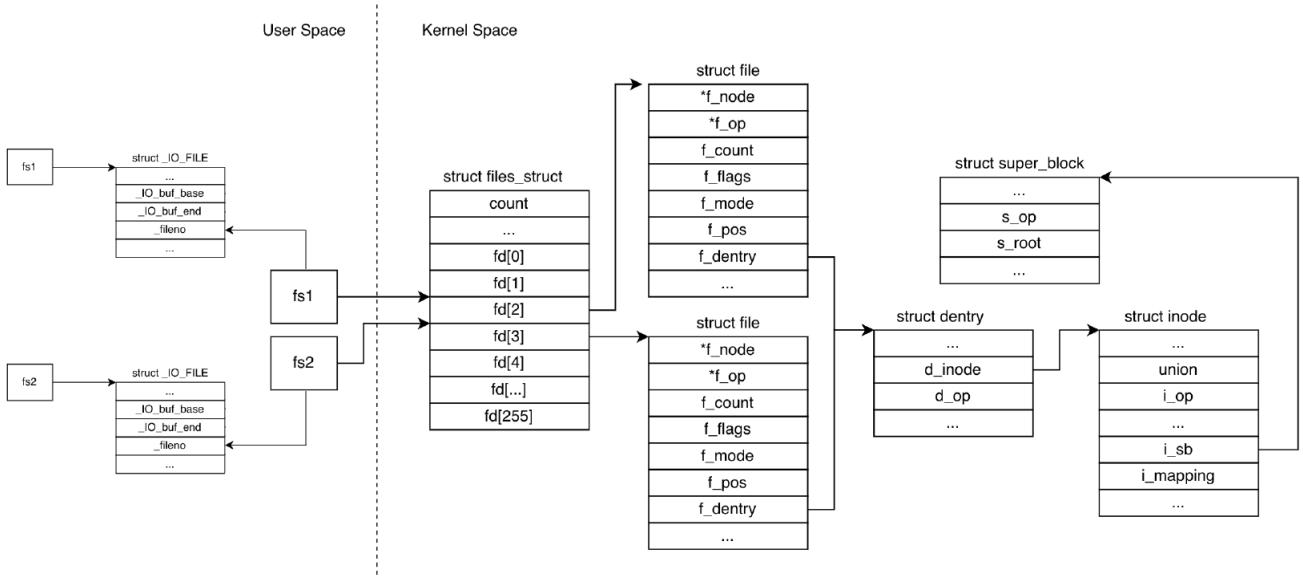


Рисунок 11 – Схема связи структур