

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

## по лабораторной работе № 5

Дисциплина: Операционные системы

Преподаватель	<div style="border-top: 1px solid black; display: inline-block; width: 150px;"></div> Н.Ю. Рязанова (И.О. Фамилия)
---------------	-----------------------------------------------------------------------------------------------------------------------

Москва, 2021

# Структура FILE ( \_IO\_FILE)

```
struct _IO_FILE;

/* The opaque type of streams. This is the definition used elsewhere. */
typedef struct _IO_FILE FILE;

...

struct _IO_FILE {
    int _flags;                /* High-order word is _IO_MAGIC; rest is flags. */
#define _IO_file_flags _flags

    /* The following pointers correspond to the C++ streambuf protocol. */
    /* Note: Tk uses the _IO_read_ptr and _IO_read_end fields directly. */
    char* _IO_read_ptr;        /* Current read pointer */
    char* _IO_read_end;        /* End of get area. */
    char* _IO_read_base;       /* Start of putback+get area. */
    char* _IO_write_base;      /* Start of put area. */
    char* _IO_write_ptr;       /* Current put pointer. */
    char* _IO_write_end;       /* End of put area. */
    char* _IO_buf_base;        /* Start of reserve area. */
    char* _IO_buf_end;         /* End of reserve area. */
    /* The following fields are used to support backing up and undo. */
    char *_IO_save_base; /* Pointer to start of non-current get area. */
    char *_IO_backup_base; /* Pointer to first valid character of backup area */
    char *_IO_save_end; /* Pointer to end of non-current get area. */

    struct _IO_marker *_markers;

    struct _IO_FILE *_chain;

    int _fileno;
    int _flags2;
    _IO_off_t _old_offset; /* This used to be _offset but it's too small. */

#define __HAVE_COLUMN /* temporary */
    /* 1+column number of pbase(); 0 is unknown. */
    unsigned short _cur_column;
    signed char _vtable_offset;
    char _shortbuf[1];

    /* char* _save_gptr; char* _save_egptr; */

    _IO_lock_t *_lock;

    _IO_off64_t _offset;
    void *__pad1;
    void *__pad2;
    void *__pad3;
    void *__pad4;
    size_t __pad5;
    int _mode;
    /* Make sure we don't get into trouble again. */
    char _unused2[15 * sizeof(int) - 4 * sizeof(void *) - sizeof(size_t)];
};
```

## Анализ структуры

Структура содержит указатели `char*` на начало, конец и текущую позицию буфера ввода, вывода и неактивного буфера. Поле `_fileno` хранит `int` дескриптор открытого файла. Также содержатся поля флагов и прав доступа.

## Задание 1

Листинг программы (однопоточная версия):

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

#define FNAME  "alph.txt"

int main()
{
    int fd = open(FNAME,O_RDONLY);
    if (fd == -1)
    {
        printf("Open failed\n");
        return -1;
    }

    FILE *fs1 = fdopen(fd,"r");
    char buff1[20];
    setvbuf(fs1,buff1,_IOFBF,20);

    FILE *fs2 = fdopen(fd,"r");
    char buff2[20];
    setvbuf(fs2,buff2,_IOFBF,20);

    int flag1 = 1, flag2 = 1;
    while(flag1 == 1 || flag2 == 1)
    {
        char c;

        flag1 = fscanf(fs1,"%c",&c);
        if (flag1 == 1)
            fprintf(stdout,"%c",c);

        flag2 = fscanf(fs2,"%c",&c);
        if (flag2 == 1)
            fprintf(stdout,"%c",c);
    }

    printf("\n");

    return 0;
}
```

Результат работы:

```
vsevolod@vsevolod-HP-Pavilion14:~/work/OS_bmstu/lab5$ gcc -o main1.o main1.c
vsevolod@vsevolod-HP-Pavilion14:~/work/OS_bmstu/lab5$ ./main1.o
aubvcwdxeyfzghijklmnopqrst
```

Буфер структуры `fs1` при первом вызове `fscanf` заполняется символами “ab...t“, буфер `fs2` заполняется оставшимся содержимым файла: “uv...z“. В каждой итерации цикла с помощью `fscanf` производится чтение из буфера `fs1` и

fs2 и запись в stdout с помощью fprintf. Поэтому в начале работы программы символы из двух буферов идут вперемешку. После считывания “z” из fs2 символы остаются только в fs1, поэтому на всех оставшихся итерациях выводятся символы только из него.

#### Листинг программы (многопоточная версия):

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

#define FNAME "alph.txt"

void* thread_f(void *data)
{
    int fd = *((int*)data);
    FILE *fs2 = fdopen(fd, "r");
    char buff2[20];
    setvbuf(fs2, buff2, _IOFBF, 20);

    int flag = 1;
    while (flag == 1)
    {
        char c;
        flag = fscanf(fs2, "%c", &c);
        if (flag == 1)
            fprintf(stdout, "%c", c);
    }
}

int main()
{
    int fd = open(FNAME, O_RDONLY);
    if (fd == -1)
    {
        printf("Open failed\n");
        return -1;
    }

    pthread_t tid;
    int err = pthread_create(&tid, NULL, thread_f, (void*)&fd);
    if (err)
    {
        printf("It's impossible to create a thread");
        return -1;
    }

    FILE *fs1 = fdopen(fd, "r");
    char buff1[20];
    setvbuf(fs1, buff1, _IOFBF, 20);

    int flag1 = 1;
    while(flag1 == 1)
    {
        char c;

        flag1 = fscanf(fs1, "%c", &c);
        if (flag1 == 1)
            fprintf(stdout, "%c", c);
    }

    err = pthread_join(tid, NULL);
    if (err)
    {

```

```

    printf("It's imposible to join the thread");
    return -1;
}

printf("\n");

return 0;
}

```

### Результат работы:

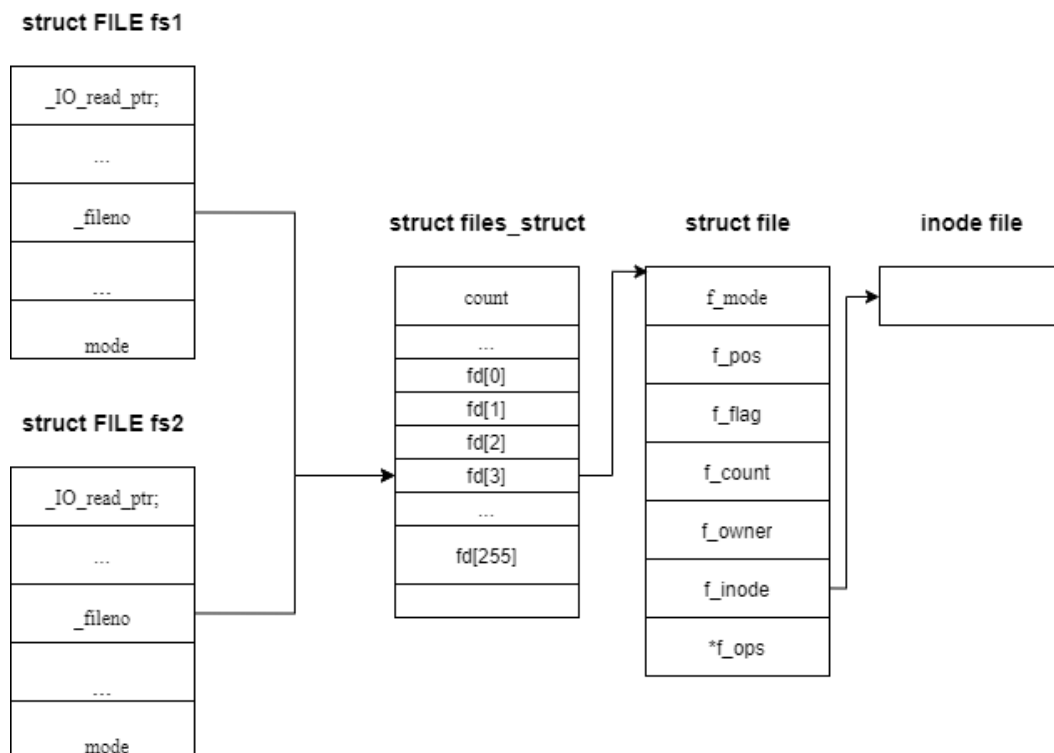
```

vsevolod@vsevolod-HP-Pavilion14:~/work/OS_bmstu/lab5$ gcc -pthread -o th1.o th1.c
vsevolod@vsevolod-HP-Pavilion14:~/work/OS_bmstu/lab5$ ./th1.o
abcdefghijklmnopqrstuvwxyz

```

В отличие от однопоточной версии, здесь содержимые буферов идут один за другим. Сначала считываются символы “ab...t” из fs1 в главном потоке, после чего “uv...z” из f2 в другом потоке. Последовательность в работе потоков связана с тем, что один из них был действующим в то время, как другой находился в состоянии сна.

### Диаграмма дескрипторов:



## Задание 2

Листинг программы (однопоточная версия):

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

#define FNAME "alph.txt"

int main()
{
    char c1, c2;

    int fd1 = open(FNAME, O_RDONLY);
    int fd2 = open(FNAME, O_RDONLY);

    if (fd1 == -1 || fd2 == -1)
    {
        printf("Open failed\n");
        return -1;
    }

    int flag1 = 1, flag2 = 1;
    while(flag1 && flag2)
    {
        flag1 = (read(fd1, &c1, 1) == 1);
        flag2 = (read(fd2, &c2, 1) == 1);
        if (flag1) write(1, &c1, 1);
        if (flag2) write(1, &c2, 1);
    }

    return 0;
}
```

Результат работы:

```
vsevolod@vsevolod-HP-Pavilion14:~/work/OS_bmstu/lab5$ gcc -o main2.o main2.c
vsevolod@vsevolod-HP-Pavilion14:~/work/OS_bmstu/lab5$ ./main2.o
aabbccddeeffgghhiijjkkllmmnnnooppqqrrssttuvwxyzvsevolod@vsevolod-HP-Pavilion14:~/work/OS_bmstu/lab5$
```

В данном случае с помощью двух вызовов `open` создаётся два разных дескриптора открытого файла "alph.txt". Поэтому `read` из `fd1` не оказывает влияния на `read` из `fd2`, т. к. они изменяют разный `f_pos`. Следовательно, `fd1` и `fd2` каждую итерацию считывают один и тот же символ из файла и выводят его с помощью `write` в `fd[1]` (т. е. `stdout`).

Листинг программы (многопоточная версия):

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

#define FNAME "alph.txt"

void* thread_f (void *data)
{
    int fd2 = open(FNAME, O_RDONLY);
    if (fd2 == -1)
        return (void*)-1;
}
```

```

char c2;
int flag2 = 1;
while(flag2)
{
    flag2 = (read(fd2,&c2,1) == 1);
    if (flag2) write(1,&c2,1);
}

return (void*)0;
}

int main()
{
    char c1;

    pthread_t tid;
    int err = pthread_create(&tid, NULL, thread_f, NULL);
    if (err)
    {
        printf("It's imposible to create a thread");
        return -1;
    }

    int fd1 = open(FNAME,O_RDONLY);
    if (fd1 == -1)
    {
        printf("Open failed\n");
        return -1;
    }

    int flag1 = 1;
    while(flag1)
    {
        flag1 = (read(fd1,&c1,1) == 1);
        if (flag1) write(1,&c1,1);
    }

    int thread_code;
    err = pthread_join(tid, (void**>(&thread_code));
    if (err)
    {
        printf("It's imposible to join the thread");
        return -1;
    }
    if (thread_code == -1)
    {
        printf("Open failed (in thread)\n");
        return -1;
    }

    return 0;
}

```

### Результат работы:

```

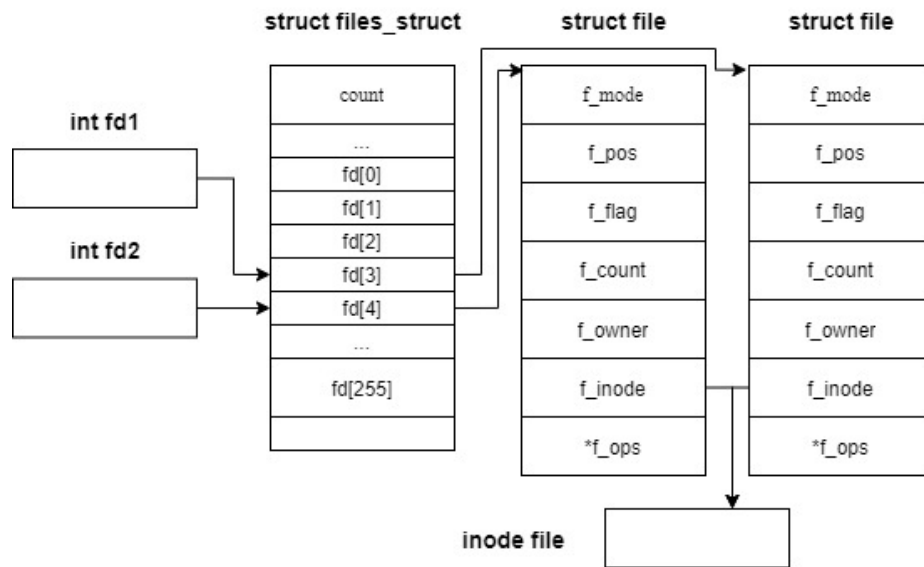
vsevolod@vsevolod-HP-Pavilion14:~/work/OS_bmstu/lab5$ gcc -pthread -o th2.o th2.c
vsevolod@vsevolod-HP-Pavilion14:~/work/OS_bmstu/lab5$ ./th2.o
abcadbecfgdheifjgkhlinjnkolpmqnrosptqurvswtxyvzwxyzvsevolod@vsevolod-HP-Pavilion14:~/work/OS_bmstu/lab5$ █

```

В данном случае, read из fd1 и fd2 происходит параллельно, поэтому символы, считанные в разных потоках выводятся вперемешку, но не стого по очереди, как в однопоточной реализации. Один из потоков в начале считывает 3

символа до того, как начинается чтение другой, поэтому возникает смещение в считываемых символах. Также существуют моменты, когда один из потоков успевает сделать два чтения до того, как это сделает другой поток.

Диаграмма дескрипторов:





## Задание 3

Листинг программы (однопоточная версия):

```
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/stat.h>

#define FNAME "out.txt"

int main()
{
    struct stat statbuf;

    FILE* fd1 = fopen(FNAME, "w");
    if (!fd1)
    {
        printf("Fopen failed\n");
        return -1;
    }
    stat(FNAME, &statbuf);
    printf("%ld, %lu\n", statbuf.st_size, statbuf.st_ino);

    FILE* fd2 = fopen(FNAME, "w");
    if (!fd2)
    {
        printf("Fopen failed\n");
    }
    stat(FNAME, &statbuf);
    printf("%ld, %lu\n", statbuf.st_size, statbuf.st_ino);

    for (int i=0; i<26; i++)
    {
        if (i % 2)
            fprintf(fd2, "%c", 'a' + i);
        else
            fprintf(fd1, "%c", 'a' + i);
    }

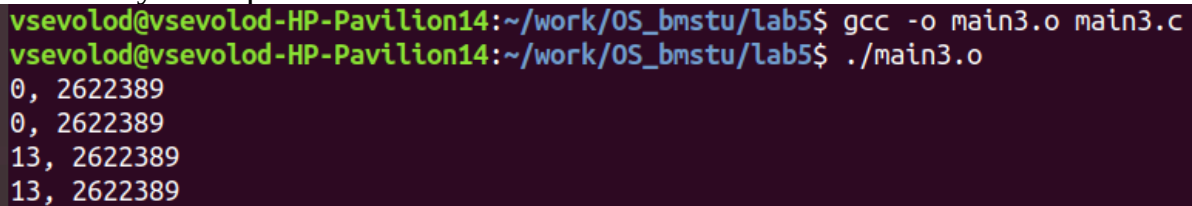
    fclose(fd2);
    stat(FNAME, &statbuf);
    printf("%ld, %lu\n", statbuf.st_size, statbuf.st_ino);

    fclose(fd1);
    stat(FNAME, &statbuf);
    printf("%ld, %lu\n", statbuf.st_size, statbuf.st_ino);

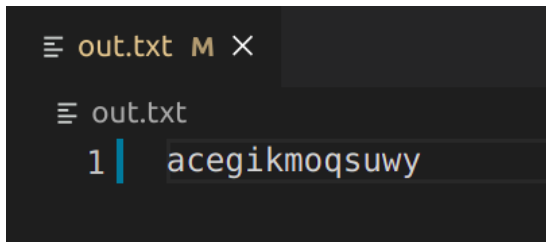
    return 0;
}

fclose(fd1);
return -1;
```

Результат работы:



```
vsevolod@vsevolod-HP-Pavilion14:~/work/OS_bmstu/lab5$ gcc -o main3.o main3.c
vsevolod@vsevolod-HP-Pavilion14:~/work/OS_bmstu/lab5$ ./main3.o
0, 2622389
0, 2622389
13, 2622389
13, 2622389
```



```
out.txt M X
out.txt
1 | acegikmoqsuwy
```

Как видно по содержимому файла, сохранился результат записи только одного из дескрипторов (для которого `fclose` был вызван последним). При `fork` происходит обнуление размера файла, его `inode` не изменяется. `fclose` также не меняет размера файла, изменяется только размер (на величину равную количеству символов в буфере).

### Листинг программы (многопоточная версия):

```
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/stat.h>
#include <pthread.h>

#define FNAME "out.txt"

void* thread_f(void *data)
{
    struct stat statbuf;
    FILE* fd2 = fopen(FNAME, "w");
    if (!fd2)
    {
        printf("Fopen failed\n");
        return (void*)-1;
    }
    stat(FNAME, &statbuf);
    printf("%ld, %lu\n", statbuf.st_size, statbuf.st_ino);

    for (int i=1; i<26; i+=2)
        fprintf(fd2, "%c", 'a' + i);

    fclose(fd2);
    stat(FNAME, &statbuf);
    printf("%ld, %lu\n", statbuf.st_size, statbuf.st_ino);

    return (void*)0;
}

int main()
{
    pthread_t tid;
    int err = pthread_create(&tid, NULL, thread_f, NULL);
    if (err)
    {
        printf("It's imposible to create a thread");
        return -1;
    }

    ///
    struct stat statbuf;
    FILE* fd1 = fopen(FNAME, "w");
    if (!fd1)
```

```

{
    printf("Fopen failed\n");
    return -1;
}
stat(FNAME, &statbuf);
printf("%ld, %lu\n", statbuf.st_size, statbuf.st_ino);

for (int i=0; i<26; i+=2)
    fprintf(fd1, "%c", 'a' + i);

fclose(fd1);
stat(FNAME, &statbuf);
printf("%ld, %lu\n", statbuf.st_size, statbuf.st_ino);

///
int thread_code;
err = pthread_join(tid, (void*)&thread_code);
if (err)
{
    printf("It's imposible to join the thread");
    return -1;
}
if (thread_code == -1)
{
    printf("Open failed (in thread)\n");
    return -1;
}

return 0;
}

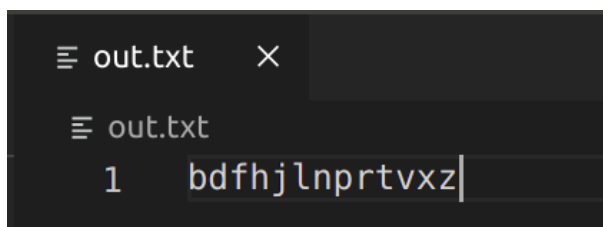
```

### Результат работы:

```

vsevolod@vsevolod-HP-Pavilion14:~/work/OS_bmstu/lab5$ gcc -pthread -o th3.o th3.c
vsevolod@vsevolod-HP-Pavilion14:~/work/OS_bmstu/lab5$ ./th3.o
0, 2622389
0, 2622389
13, 2622389
13, 2622389
vsevolod@vsevolod-HP-Pavilion14:~/work/OS_bmstu/lab5$

```



В многопоточной версии наблюдается такое же изменение inode и размера файла, как было описано выше. Содержимое файла зависит от того, какой из потоков сделает `fclose` первым, что неоднозначно, поэтому в результирующем файле оказываются как чётные, так и нечётные символы.

### Диаграмма дескрипторов:

**struct FILE fs1**

_IO_read_ptr;
...
_fileno
...
mode

**struct FILE fs2**

_IO_read_ptr;
...
_fileno
...
mode

**struct files\_struct**

count
...
fd[0]
fd[1]
fd[2]
fd[3]
fd[4]
...
fd[255]

**struct file**

f_mode
f_pos
f_flag
f_count
f_owner
f_inode
*f_ops

**struct file**

f_mode
f_pos
f_flag
f_count
f_owner
f_inode
*f_ops

inode file

