



**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»

**Отчёт
к лабораторной работе № 5
По курсу: «Операционные системы»**

Студент Мередова А.

Группа ИУ7-66Б

Оценка (баллы) _____

Преподаватели **Рязанова Н. Ю.**

Москва.
2021 г.

Задание

В лабораторной работе анализируется результат выполнения трех программ. Программы демонстрируют открытие одного и того же файла несколько раз. Реализация открытия файла в одной программе несколько раз выбрана для простоты. Такая ситуация возможна в системе, когда один и тот же файл несколько раз открывают разные процессы. Но для получения ситуаций аналогичных тем, которые демонстрируют приведенные программы надо было бы синхронизировать работу процессов. При выполнении асинхронных процессов такая ситуация вероятна и ее надо учитывать, чтобы избежать потери данных или получения неверного результата при выводе в файл.

Структура FILE

stdio.h

```
typedef struct __sFILE {
    unsigned char *_p; /* current position in (some) buffer */
    int _r; /* read space left for getc() */
    int _w; /* write space left for putc() */
    short _flags; /* flags, below; this FILE is free if 0 */
    short _file; /* fileno, if Unix descriptor, else -1 */
    struct __sbuf _bf; /* the buffer (at least 1 byte, if !NULL) */
    int _lbfsize; /* 0 or -_bf._size, for inline putc */

    /* operations */stdio.h
    void *_cookie; /* cookie passed to io functions */
    int (*_Nullable _close)(void *);
    int (*_Nullable _read)(void *, char *, int);
    fpos_t(*_Nullable _seek)(void *, fpos_t, int);
    int (*_Nullable _write)(void *, const char *, int);

    /* separate buffer for long sequences of ungetc() */
    struct __sbuf _ub; /* ungetc buffer */
    struct __sFILEX *_extra; /* additions to FILE to not break ABI */
    int _ur; /* saved _r when _r is counting ungetc data */

    /* tricks to meet minimum requirements even when malloc() fails */
    unsigned char _ubuf[3]; /* guarantee an ungetc() buffer */
    unsigned char _nbuf[1]; /* guarantee a getc() buffer */

    /* separate buffer for fgetln() when line crosses buffer boundary */
    struct __sbuf _lb; /* buffer for fgetln() */
}
```

```
/* Unix stdio files get aligned to block boundaries on fseek() */
int  _blksize; /* stat.st_blksize (may be != _bf._size) */
fpos_t_offset; /* current lseek offset (see WARNING) */
} FILE;
```

Проанализировать работу приведенных программ и объяснить результаты их работы.

Первая программа:

```
#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);

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

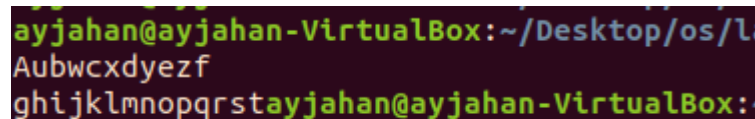
    int flag1 = 1, flag2 = 2;
    while (flag1 == 1 || flag2 == 1)
    {
        char c;
        flag1 = fscanf(fs1, "%c", &c);
        if (flag1 == 1)
            fprintf(stdout, "%c", c);

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

    }

    return 0;
}
```

Результат работы программы



```
ayjahan@ayjahan-VirtualBox:~/Desktop/os/l
Aubwxc dyezf
ghijklmnopqrstayjahan@ayjahan-VirtualBox:
```

Анализ работы программы

С помощью системного вызова `open()` создается дескриптор открытого файла. Системный вызов `open()` возвращает индекс в массиве `fd` структуры `files_struct`.

Вызов `fdopen()` создает структуры типа `FILE(fs1 и fs2)`, которые ссылаются на дескриптор открытого файла, созданный системным вызовом `open`.

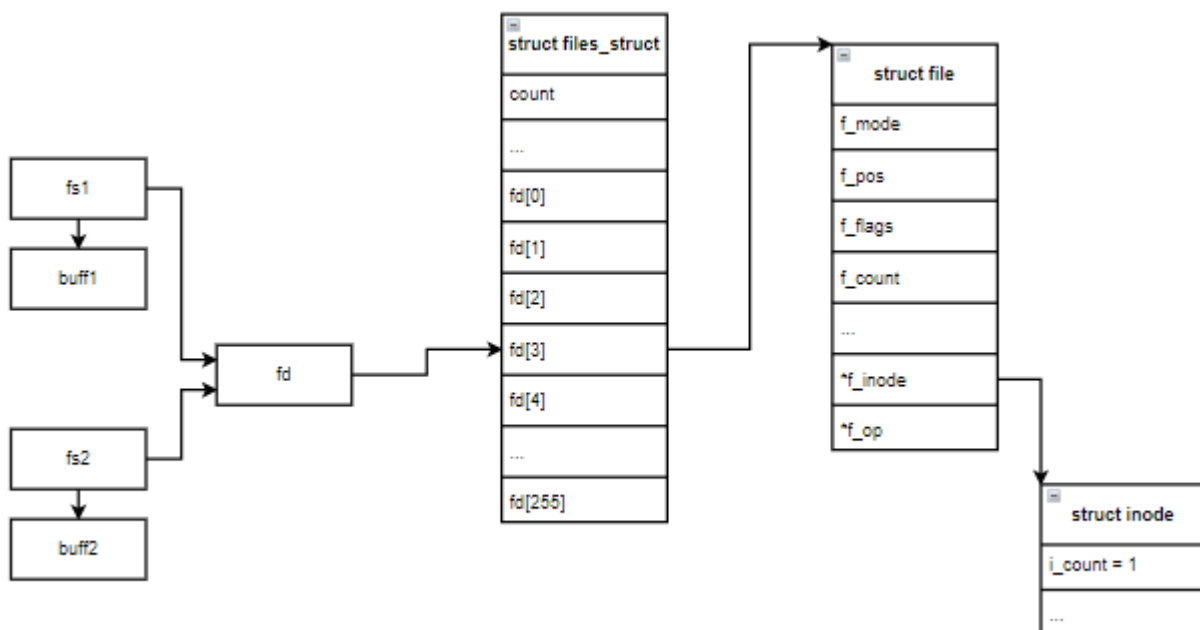
Для дескрипторов `fs1` и `fs2` помощью `setbuf` задаём соответствующие буферы и тип буферизации `_IOFBF (INPUT OUTPUT FULL)`.

Выполняем в цикле `fscanf()` для `fs1` и `fs2` поочередно.

Так как установлен тип буферизации `_IOFBF` (полная), то буфер будет заполнен при первом вызове `fscanf()`, указатель `f_pos` установится на следующий за последним записанным в буфер символ.

При первом вызове `fscanf(fs1,"%c",&c)` в буфер `buff1` считываются первые 20 символов (с а по t включительно). При первом вызове `fscanf(fs2,"%c",&c)`, в буфер `buff2` считываются оставшиеся в файле символы – (с и по z). При дальнейших вызовах в цикле будут поочередно выводиться символы из `buff1` и `buff2` до тех пор, пока символы в одном из буферов не закончатся. Тогда на экран будут последовательно выведены оставшиеся символы из другого буфера.

Схема связей структур



Вторая программа (без доп потоков)

```
#include <fcntl.h>

int main()
{
    char c;
    int fd1 = open("alphabet.txt", O_RDONLY);
    int fd2 = open("alphabet.txt", O_RDONLY);

    while (read(fd1, &c, 1) == 1 && read(fd2, &c, 1) == 1)
    {
        write(1, &c, 1);
        write(1, &c, 1);
    }
    return 0;
}
```

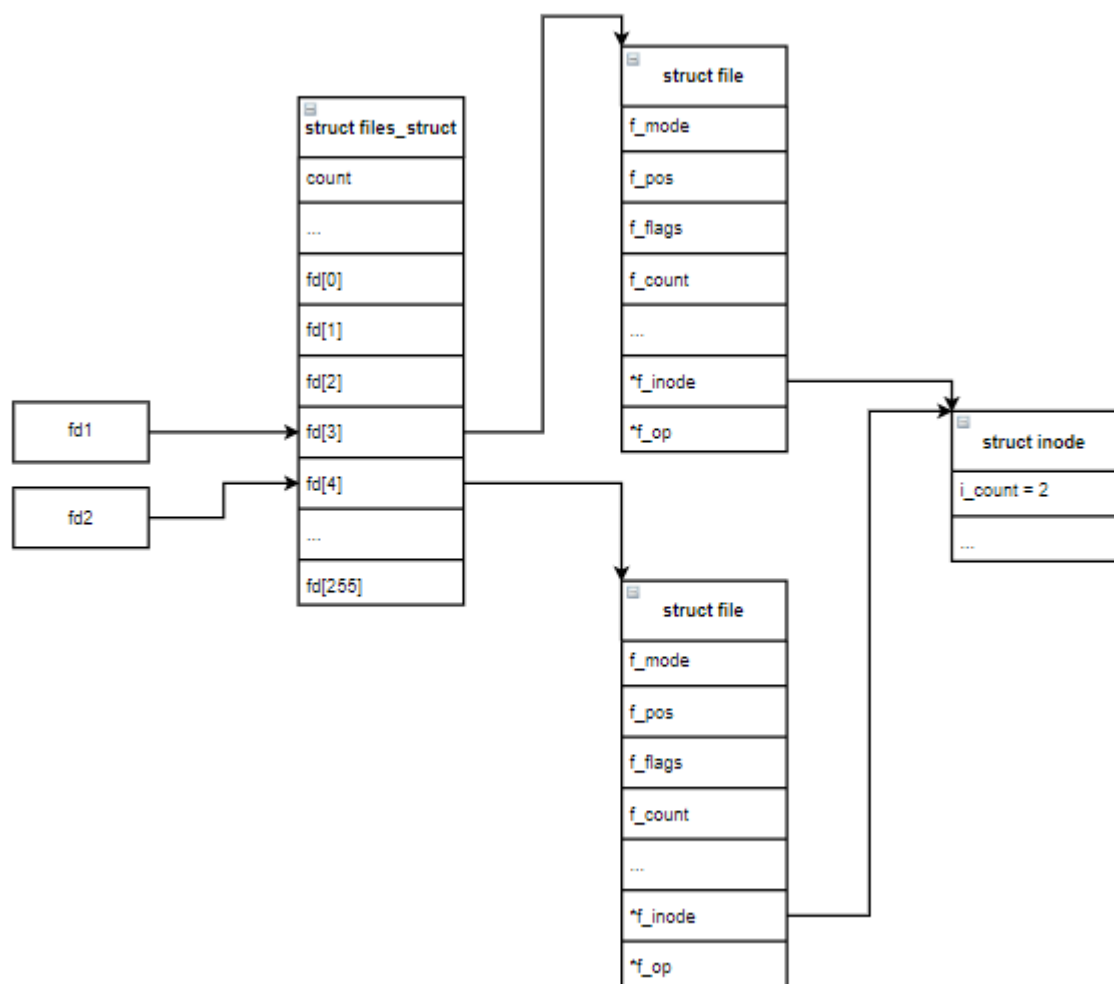
Результат работы программы:

```
ayjahan@ayjahan-VirtualBox:~/Desktop/os/lab05$ ./2
AAbbccddeeffgghhiijjkkllmmnnnooppqrrssttuuvvwwxxyyzz
```

Анализ работы программы:

Один и тот же файл "alphabet.txt" открыт 2 раза для чтения. При вызове `open()` создается дескриптор открытого файла в системной таблице открытых файлов и дескриптор файла в таблице файлов, открытых процессом. В данной программе после вызова `open()` в таблице открытых файлов будет 2 дескриптора, каждый из которых содержит собственный указатель `f_pos`. При вызове `read()` для дескрипторов их указатели `f_pos` сдвигаются по файлу независимо от указателя другого дескриптора, таким образом каждый символ считывается и выводится по два раза. Двум дескрипторам открытого файла соответствует один `inode`.

Схема связей структур:



Вторая программа (с доп потоками)

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

void *read_file(int *fd)
{
    char c;
    while (read(*fd, &c, 1) == 1)
    {
        write(1, &c, 1);
    }

    sleep(rand() % 2);
}

int main()
{
    int fd1 = open("alph.txt", O_RDONLY);
    int fd2 = open("alph.txt", O_RDONLY);

    pthread_t thread1;
```

```

pthread_t thread2;

int status1 = pthread_create(&thread1, NULL, read_file, &fd1);
if (status1 != 0)
{
    printf("Error: can't create thread 1\n");
    return -1;
}

int status2 = pthread_create(&thread2, NULL, read_file, &fd2);
if (status2 != 0)
{
    printf("Error: can't create thread 2\n", status2);
    return -1;
}

pthread_join(thread1, NULL);
pthread_join(thread2, NULL);
return 0;
}

```

Результат работы программы:

```

ayjahan@ayjahan-VirtualBox:~/Desktop/os/lab05$ ./2
AAbbccddeeffgghhiijjkkllmmnnooppqrrssttuuvvwwxxyyzz

```

Третья программа (без доп потоков):

```

#include <stdio.h>
#include <sys/stat.h>
#include <errno.h>
int main()
{
    struct stat statbuf;

    FILE *fs1 = fopen("writeres.txt", "w");
    stat("writeres.txt", &statbuf);
    printf(" + FOPEN FS1: inode = %ld, buffsize = %ld, blocksize= %ld\n",
        (long int)statbuf.st_ino,
        (long int)statbuf.st_size,
        (long int)statbuf.st_blksize);

    FILE *fs2 = fopen("writeres.txt", "w");
    stat("writeres.txt", &statbuf);
    printf(" + FOPEN FS2: inode = %ld, buffsize = %ld, blocksize= %ld\n",
        (long int)statbuf.st_ino,
        (long int)statbuf.st_size,
        (long int)statbuf.st_blksize);

    for (char c = 'a'; c <= 'z'; c++)
    {

```

```

        if (c % 2)
            fprintf(fs1, "%c", c);
        else
            fprintf(fs2, "%c", c);
    }

    fclose(fs1);
    stat("writeres.txt", &statbuf);
    printf(" + FCLOSE FS1: inode = %ld, buffsize = %ld, blocksize= %ld\n",
        (long int)statbuf.st_ino,
        (long int)statbuf.st_size,
        (long int)statbuf.st_blksize);

    fclose(fs2);
    stat("writeres.txt", &statbuf);
    printf(" + FCLOSE FS2: inode = %ld, buffsize = %ld, blocksize= %ld\n",
        (long int)statbuf.st_ino,
        (long int)statbuf.st_size,
        (long int)statbuf.st_blksize);

    return 0;
}

```

Результат работы программы:

```

ayjahan@ayjahan-VirtualBox:~/Desktop/os/lab05$ gcc 3.c -w -pthread -o 3
ayjahan@ayjahan-VirtualBox:~/Desktop/os/lab05$ ./3
+ FOPEN FS1: inode = 272771, buffsize = 0, blocksize= 4096
+ FOPEN FS2: inode = 272771, buffsize = 0, blocksize= 4096
+ FCLOSE FS1: inode = 272771, buffsize = 13, blocksize= 4096
+ FCLOSE FS2: inode = 272771, buffsize = 13, blocksize= 4096
ayjahan@ayjahan-VirtualBox:~/Desktop/os/lab05$ cat writeres.txt
bdfhjlnprtvxayjahan@ayjahan-VirtualBox:~/Desktop/os/lab05$

```

Другой порядок вызовов fclose()

```

#include <sys/stat.h>
#include <errno.h>
int main()
{
    struct stat statbuf;

    FILE *fs1 = fopen("writeres.txt", "w");
    stat("writeres.txt", &statbuf);
    printf(" + FOPEN FS1: inode = %ld, buffsize = %ld, blocksize= %ld\n",
        (long int)statbuf.st_ino,
        (long int)statbuf.st_size,

```



```

        (long int)statbuf.st_blksize);

FILE *fs2 = fopen("writeres.txt", "w");
stat("writeres.txt", &statbuf);
printf(" + FOPEN FS2: inode = %ld, buffsize = %ld, blocksize= %ld\n",
        (long int)statbuf.st_ino,
        (long int)statbuf.st_size,
        (long int)statbuf.st_blksize);

for (char c = 'a'; c <= 'z'; c++)
{
    if (c % 2)
        fprintf(fs1, "%c", c); //aceg...
    else
        fprintf(fs2, "%c", c); //bdfh...
}

fclose(fs2);
stat("writeres.txt", &statbuf);
printf(" + FCLOSE FS2: inode = %ld, buffsize = %ld, blocksize= %ld\n",
        (long int)statbuf.st_ino,
        (long int)statbuf.st_size,
        (long int)statbuf.st_blksize);

fclose(fs1);
stat("writeres.txt", &statbuf);
printf(" + FCLOSE FS1: inode = %ld, buffsize = %ld, blocksize= %ld\n",
        (long int)statbuf.st_ino,
        (long int)statbuf.st_size,
        (long int)statbuf.st_blksize);

return 0;
}

```

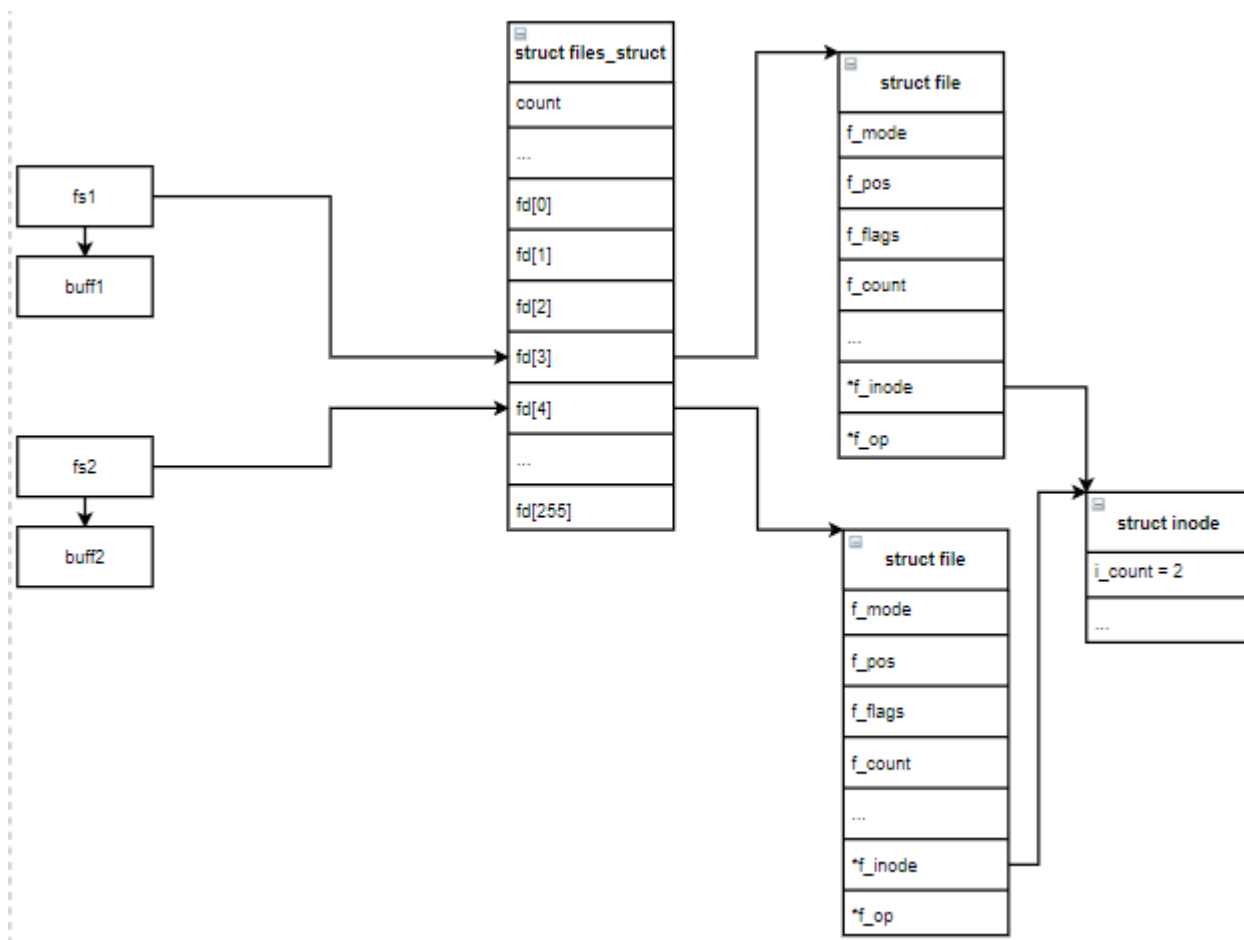
Результат работы программы:

```

ayjahan@ayjahan-VirtualBox:~/Desktop/os/lab05$ ./3_2
+ FOPEN FS2: inode = 272771, buffsize = 0 blocksize= 4096
+ FOPEN FS1: inode = 272771, buffsize = 0 blocksize= 4096
+ FCLOSE FS1: inode = 272771, buffsize = 13 blocksize= 4096
+ FCLOSE FS2: inode = 272771, buffsize = 13 blocksize= 4096
ayjahan@ayjahan-VirtualBox:~/Desktop/os/lab05$ cat writeres.txt
acegikmoqsuwyayjahan@ayjahan-VirtualBox:~/Desktop/os/lab05$

```

Схема связей структур:



Анализ работы программы:

При вызове `open` для записи создается новый файл. Создается два дескриптора открытых файлов, каждый из которых содержит позицию `f_pos`, не зависящую от второго дескриптора. Два дескриптора открытых файлов соответствуют одному и тому же `inode`.

`fprintf` – функция буферизованного ввода/вывода, т.е. сначала данные записываются в буфер. Данные выводятся из буфера в файл одним из трех случаев:

- 1) Буфер заполнен.
- 2) Вызов `fflush` - принудительная запись содержимого в файл.
- 3) Вызов `fclose`.

Если сначала вызывается `fclose(fs1)`, а затем `fclose(fs2)`:

- При вызове `fclose()` для `fs1` буфер для `fs1` записывается в файл. При вызове `fclose()` для `fs2`, все содержимое файла очищается, а в файл записывается содержимое буфера для `fs2`. **Произошла утеря данных `fs1`, в файле окажется только содержимое буфера для `fs2`.**

Если сначала вызывается `fclose(fs2)`, а затем `fclose(fs1)`:

- При вызове `fclose()` для `fs2` буфер для `fs2` записывается в файл. При вызове `fclose()` для `fs1`, все содержимое файла очищается, а в файл записывается содержимое буфера для `fs1`. **Произошла утеря данных `fs2`, в файле окажется только содержимое буфера для `fs1`.**

Третья программа с использованием потоков

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

void *write_to_file(int *filenum)
{
    struct stat statbuf;

    FILE *fs = fopen("writeres.txt", "w");
    stat("writeres.txt", &statbuf);
    printf("+ FOPEN FS%d: inode = %ld, buffersize = %ld blocksize= %ld\n",
        *filenum,
        (long int)statbuf.st_ino,
        (long int)statbuf.st_size,
        (long int)statbuf.st_blksize);

    for (char c = 'a'; c <= 'z'; c++)
    {
        if (c % 2 && *filenum == 1) //acegi...
            fprintf(fs, "%c", c);

        if (!(c % 2) && *filenum == 2) //bdfh...
            fprintf(fs, "%c", c);
    }

    fclose(fs);
    stat("writeres.txt", &statbuf);
    printf("+ FCLOSE FS%d: inode = %ld, buffersize = %ld blocksize= %ld\n",
        *filenum,
        (long int)statbuf.st_ino,
        (long int)statbuf.st_size,
        (long int)statbuf.st_blksize);
}

int main()
{
    pthread_t thread1;
    pthread_t thread2;

    int num1 = 1;
    int num2 = 2;

    if (pthread_create(&thread1, NULL, write_to_file, &num1) != 0)
```

```

{
    printf("Error: can't create thread 1\n");
    return -1;
}

if (pthread_create(&thread2, NULL, write_to_file, &num2) != 0)
{
    printf("Error: can't create thread 2\n");
    return -1;
}

pthread_join(thread1, NULL);
pthread_join(thread2, NULL);

return 0;
}

```

Демонстрация работы программы:

```

ayjahan@ayjahan-VirtualBox:~/Desktop/os/lab05$ ./3
+ FOPEN FS2: inode = 272771, buffsize = 0 blocksize= 4096
+ FCLOSE FS2: inode = 272771, buffsize = 13 blocksize= 4096
+ FOPEN FS1: inode = 272771, buffsize = 0 blocksize= 4096
+ FCLOSE FS1: inode = 272771, buffsize = 13 blocksize= 4096
ayjahan@ayjahan-VirtualBox:~/Desktop/os/lab05$ cat writeres.txt
acegikmoqsuwyayjahan@ayjahan-VirtualBox:~/Desktop/os/lab05$

```

Сначала создается поток для fs2, затем для fs1.

```

#include <stdio.h>
#include <sys/stat.h>
#include <pthread.h>

void *write_to_file(int *filenum)
{
    struct stat statbuf;

    FILE *fs = fopen("writeres.txt", "w");
    stat("writeres.txt", &statbuf);
    printf("+ FOPEN FS%d: inode = %ld, buffsize = %ld blocksize= %ld\n", *filenum
,
        (long int)statbuf.st_ino,
        (long int)statbuf.st_size,
        (long int)statbuf.st_blksize);

    for (char c = 'a'; c <= 'z'; c++)
    {
        if (c % 2 && *filenum == 1) //acegi...

```

```

        fprintf(fs, "%c", c);

        if (!(c % 2) && *filenum == 2) //bdfh...
            fprintf(fs, "%c", c);
    }

    fclose(fs);
    stat("writeres.txt", &statbuf);
    printf("+ FCLOSE FS%d: inode = %ld, buffsize = %ld blocksize= %ld\n", *filen
um,
        (long int)statbuf.st_ino,
        (long int)statbuf.st_size,
        (long int)statbuf.st_blksize);
}

int main()
{
    pthread_t thread1;
    pthread_t thread2;

    int num1 = 1;
    int num2 = 2;

    if (pthread_create(&thread2, NULL, write_to_file, &num2) != 0)
    {
        printf("Error: can't create thread 2\n");
        return -1;
    }

    if (pthread_create(&thread1, NULL, write_to_file, &num1) != 0)
    {
        printf("Error: can't create thread 1\n");
        return -1;
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}

```

Результат работы программы:

```

ayjahan@ayjahan-VirtualBox:~/Desktop/os/lab05$ ./3_2
+ FOPEN FS2: inode = 272771, buffsize = 0 blocksize= 4096
+ FOPEN FS1: inode = 272771, buffsize = 0 blocksize= 4096
+ FCLOSE FS1: inode = 272771, buffsize = 13 blocksize= 4096
+ FCLOSE FS2: inode = 272771, buffsize = 13 blocksize= 4096
ayjahan@ayjahan-VirtualBox:~/Desktop/os/lab05$ cat writeres.txt
acegikmoqsuwyayjahan@ayjahan-VirtualBox:~/Desktop/os/lab05$

```