

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04 Программная инженерия**

по лабораторной работе № 5

Дисциплина: Операционные системы

Преподаватель	<hr/>	Н.Ю. Рязанова
	(Подпись, дата)	(И.О. Фамилия)

Москва, 2020

Структура FILE (/usr/include/x86_64-linux-gnu/bits/types/FILE.h)

```
#ifndef __FILE_defined
#define __FILE_defined 1

struct _IO_FILE;

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

#endif
```

Структура _IO_FILE (/usr/include/bits/libio.h/_IO_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;
#ifdef 0
    int _blksize;
#else
    int _flags2;
#endif
    _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;  
#ifdef _IO_USE_OLD_IO_FILE  
};
```

В данной структуре содержится информация о флагах, правах доступа, текущий указатель для чтения/записи, информация о резервной зоне (буфере), номер дескриптора открытого файла и т.д.

1 задание

```
#include <stdio.h>  
#include <fcntl.h>  
#include <string.h>  
  
#define FILENAME "alphabet.txt"  
#define SIZE 20  
  
int main()  
{  
    int fd = open(FILENAME, O_RDONLY);  
    if (fd == -1)  
    {  
        printf("ERROR: can't open file %d", fd);  
        return -1;  
    }  
  
    FILE *fs1 = fdopen(fd, "r");  
    char buff1[SIZE];  
    setvbuf(fs1, buff1, _IOFBF, SIZE);  
  
    FILE *fs2 = fdopen(fd, "r");  
    char buff2[SIZE];  
    setvbuf(fs2, buff2, _IOFBF, SIZE);  
  
    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);  
    }  
  
    return 0;  
}
```

Результат работы

ekaterina@ekaterina-HP-470-G7-Notebook-PC:
aubvcwdxeyfzghijklmnopqrst

Анализ работы

И buff1, и buff2 имеют размер, равный 20 элементам. Поскольку задан параметр _IOFBF (блочная буферизация) и первым производится работа с fs1, то в buff1 попадают символы от а до t (20 элементов), а в buff2 только оставшиеся 6 от u до z. Так как в цикле производится поочередное чтение по одному символу из соответствующих fs1 и fs2 буферов, то на экран выводится сначала символ из buff1, потом из buff2. Поочередное чтение продолжается, пока полностью не будет считана информация из buff2 (так как символов меньше), далее символы будут выводиться только из buff1 до тех пор, пока есть, что читать из него.

1 задание (многопоточность)

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

#define FILENAME  "alphabet.txt"
#define SIZE      20

void* thr_fn(void *arg)
{
    int fd = *((int*)arg);
    FILE *fs2 = fdopen(fd, "r");
    char buff2[SIZE];
    setvbuf(fs2, buff2, _IOFBF, SIZE);

    int flag2 = 1;
    while (flag2 == 1)
    {
        char c;

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

    return 0;
}

int main()
{
    int fd = open(FILENAME, O_RDONLY);
    if (fd == -1)
    {
        printf("ERROR: can't open file %d", fd);
        return -1;
    }

    pthread_t tid;
    if (pthread_create(&tid, NULL, thr_fn, (void*)&fd))
    {
        printf("ERROR: mistake in creating thread");
        return -1;
    }

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

```

int flag1 = 1;

while (flag1 == 1)
{
    char c;

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

if (pthread_join(tid, NULL))
{
    printf("ERROR: mistake in joining thread");
    return -1;
}
return 0;
}

```

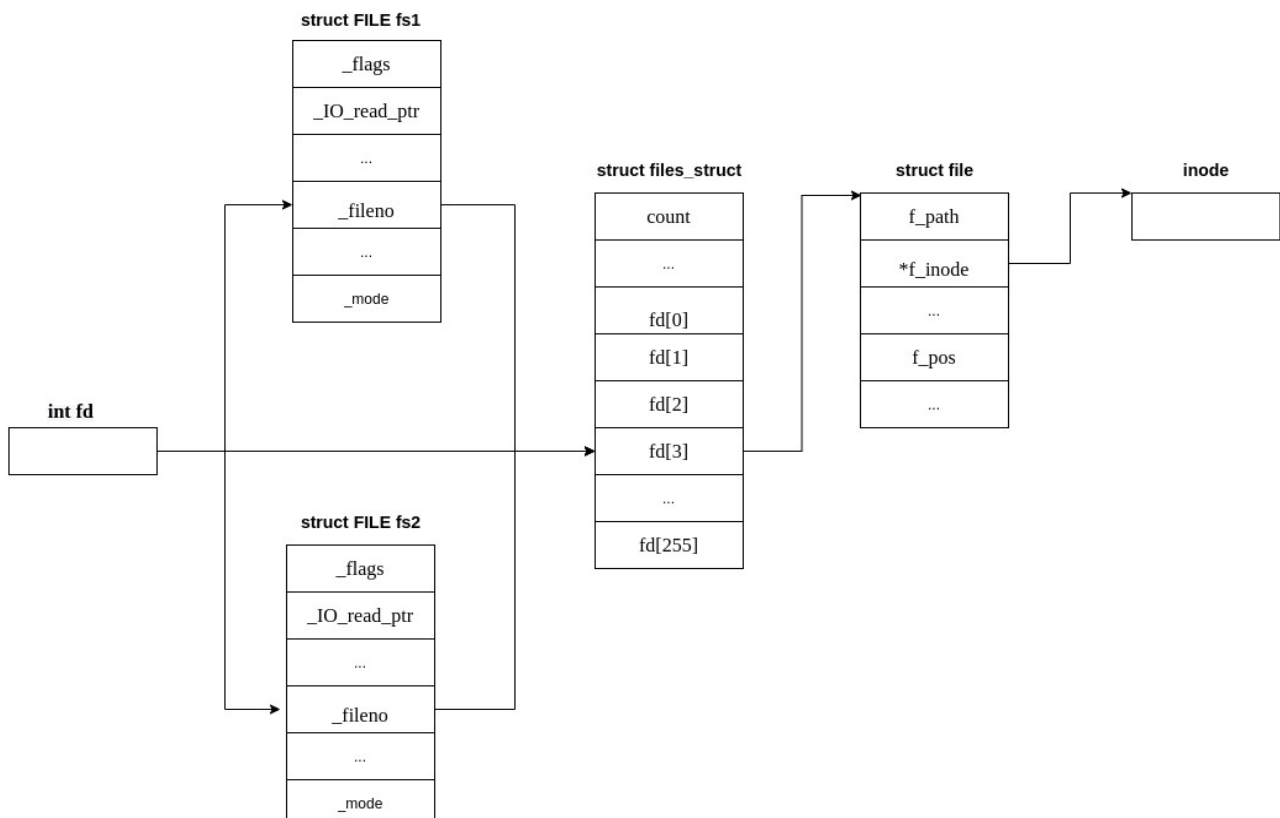
Результат работы

ekaterina@ekaterina-HP-470-G7-Notebook-PC
 abcdefghijklmnopqrstuvwxyz

Анализ работы

Это многопоточная версия программы. Buff1 и buff2 также имеют одинаковый размер в 20 элементов. Результат работы программы отличается тем, что символы выводятся не поочередно, как было раньше, а блоками, которые были записаны в buff1 и buff2.

Схема:



2 задание

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

#define FILENAME "alphabet.txt"

int main()
{
    char c;
    int fd1 = open(FILENAME, O_RDONLY);
    if (fd1 == -1)
    {
        printf("ERROR: can't open file %d", fd1);
        return -1;
    }

    int fd2 = open(FILENAME, O_RDONLY);
    if (fd2 == -1)
    {
        printf("ERROR: can't open file %d", fd2);
        return -1;
    }

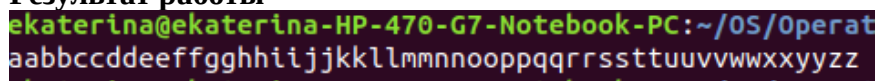
    int flag1 = 1, flag2 = 1;

    while(flag1 && flag2)
    {
        flag1 = read(fd1, &c, 1);
        if (flag1)
            write(1, &c, 1);

        flag2 = read(fd2, &c, 1);
        if (flag2)
            write(1, &c, 1);
    }

    return 0;
}
```

Результат работы



```
ekaterina@ekaterina-HP-470-G7-Notebook-PC:~/OS/Operat
aabbccddeeffgghhiijjkkllmmnnnooppqqrrssttuuvvwwxxyyzz
```

Анализ работы

Используется два системных выхода `open()`, каждый из которых создает новый файловый дескриптор открытого файла и запись в таблице открытых файлов. Таким образом, получается два файловых дескриптора файла «alphabet.txt», и в обоих есть поле `f_pos` — текущая позиция чтения/записи (в нашем случае, чтения), которая указывает на начало файла. Поэтому в процессе считывания по одному символу с помощью `read()` на экран каждый символ выводится по 2 раза, поскольку `f_pos` меняются независимо друг от друга.

2 задание (многопоточность)

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

#define FILENAME "alphabet.txt"

void* thr_fn(void *arg)
{
    char c;
    int err = *((int*)arg);
    int fd2 = open(FILENAME, O_RDONLY);
    if (fd2 == -1)
    {
        printf("ERROR: can't open file %d", fd2);
        err = -1;
        return 0;
    }

    int flag2 = 1;
    while(flag2)
    {
        flag2 = read(fd2,&c,1);
        if (flag2)
            write(1,&c,1);
    }
    return 0;
}

int main()
{
    char c;
    int err = 0;

    pthread_t tid;
    if (pthread_create(&tid, NULL, thr_fn, (void*)&err))
    {
        printf("ERROR: in creating thread");
        return -1;
    }

    int fd1 = open(FILENAME, O_RDONLY);
    if (fd1 == -1)
    {
        printf("ERROR: can't open file %d", fd1);
        return -1;
    }

    int flag1 = 1;
    while(flag1)
    {
```

```

    flag1 = read(fd1,&c,1);
    if (flag1)
        write(1,&c,1);
}

if (pthread_join(tid, NULL))
{
    printf("ERROR: in joining thread");
    return -1;
}
if (err)
{
    printf("ERROR: in reading in thread");
    return -1;
}
return 0;
}

```

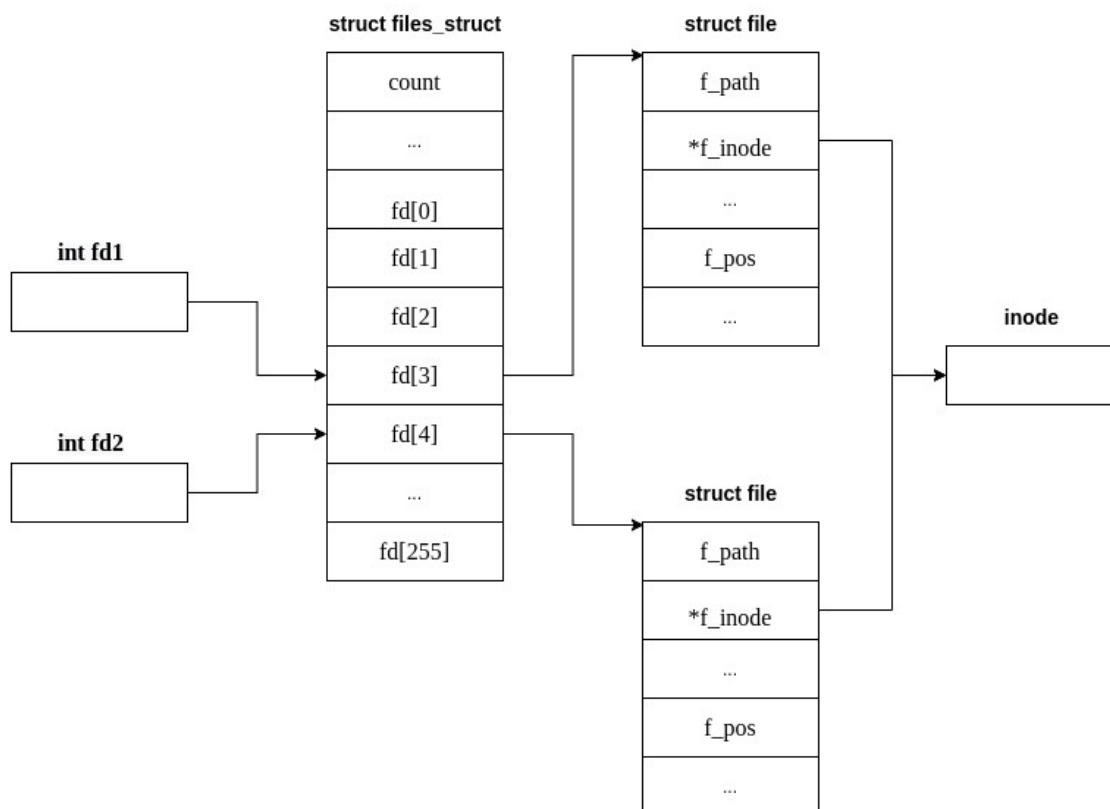
Результат работы

eka
 t
 e
 r
 i
 n
 a
 g
 e
 k
 a
 t
 e
 r
 i
 n
 a
 -
 H
 P
 -
 4
 7
 0
 -
 G
 7
 -
 N
 o
 t
 e
 b
 o
 o
 k
 -
 P
 C
 :
 ~
 /
 O
 S
 /
 O
 p
 e
 r
 a
 t
 i
 n
 g
 s
 y
 s
 t
 e
 m
 s
 a
 n
 d
 a
 p
 p
 l
 i
 c
 a
 t
 i
 o
 n
 s
 a
 b
 c
 d
 e
 f
 g
 a
 h
 b
 i
 c
 j
 d
 k
 e
 l
 f
 m
 g
 n
 h
 o
 i
 p
 j
 q
 r
 l
 s
 m
 t
 n
 u
 o
 v
 p
 w
 q
 x
 r
 y
 s
 z
 t
 u
 v
 w
 x
 y
 z

Анализ работы

Многопоточная версия программы. Символы также дублируются, только теперь второй раз символ выводится не сразу, так как не гарантируется, что, после того, как один поток выведет на экран символ, второй после этого сделает тоже самое.

Схема



3 задание

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

#define FILENAME "alphabet_3.txt"

int main()
{
    struct stat sbuf;
    char c[] = "abcdefghijklmnopqrstuvwxyzABC";

    FILE* fd1 = fopen(FILENAME, "w");
    if (!fd1)
    {
        printf("ERROR: can't open file for 1st time");
        return -1;
    }
    stat(FILENAME, &sbuf);
    printf("\n>>> inode: %d\t size: %d\n", (int)sbuf.st_ino, (int)sbuf.st_size);

    FILE* fd2 = fopen(FILENAME, "w");
    if (!fd2)
    {
        printf("ERROR: can't open file for 2nd time");
        return -1;
    }
    stat(FILENAME, &sbuf);
    printf("\n>>> inode: %d\t size: %d\n", (int)sbuf.st_ino, (int)sbuf.st_size);

    int i = 0;

    while(i < strlen(c))
    {
        if (i % 2)
            fprintf(fd2, "%c", c[i]);
        else
            fprintf(fd1, "%c", c[i]);

        i += 1;
    }

    fclose(fd2);
    stat(FILENAME, &sbuf);
    printf("\n>>> inode: %d\t size: %d\n", (int)sbuf.st_ino, (int)sbuf.st_size);

    fclose(fd1);
    stat(FILENAME, &sbuf);
    printf("\n>>> inode: %d\t size: %d\n", (int)sbuf.st_ino, (int)sbuf.st_size);
```

```
return 0;
}
```

Результат работы

```
>>> inode: 1313305      size: 0
>>> inode: 1313305      size: 0
>>> inode: 1313305      size: 14
>>> inode: 1313305      size: 15
```

acegikmoqsuwyzAC

Анализ работы

Используется два раза библиотечная функция `fopen()`, объявляется два файловых дескриптора `fd1`, `fd2`. Символы поочередно записываются то в один, то в другой, при вызове `fclose()` содержимое переписывается в файл «`alphabet_3.txt`». Сначала функция `fclose()` применяется к `fd2`, поэтому размер файла «`alphabet_3.txt`» равен 14 байтам, затем к `fd1`, при этом существующая в файле информация теряется и записываются буквы, относящиеся к `fd1`. Так как количество символов больше прежнего на 1, то и соответственно размер файла также увеличился, при этом `inode` не менялся на протяжении всей работы программы.

3 задание (многопоточность)

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

#define FILENAME  "alphabet_3.txt"

char c[] = "abcdefghijklmnopqrstuvwxyzABC";

void* thr_fn(void *arg)
{
    int err = *((int*)arg);
    struct stat sbuf;

    FILE* fd2 = fopen(FILENAME, "w");
    if (!fd2)
    {
        printf("ERROR: can't open file for 2nd time");
        err = -1;
        return 0;
    }
    stat(FILENAME, &sbuf);
    printf("\n>>> inode: %d\t size: %d\n", (int)sbuf.st_ino, (int)sbuf.st_size);

    int i = 0;
```

```

while(i < strlen(c))
{
    fprintf(fd2, "%c", c[i]);
    i += 2;
}

fclose(fd2);
stat(FILENAME, &sbuf);
printf("\n>>> inode: %d\t size: %d\n", (int)sbuf.st_ino, (int)sbuf.st_size);

return 0;
}

int main()
{
    struct stat sbuf;
    int err = 0;

    pthread_t tid;
    if (pthread_create(&tid, NULL, thr_fn, (void*)&err))
    {
        printf("ERROR: in creating thread");
        return -1;
    }

    FILE* fd1 = fopen(FILENAME, "w");
    if (!fd1)
    {
        printf("ERROR: can't open file for 1st time");
        return -1;
    }
    stat(FILENAME, &sbuf);
    printf("\n>>> inode: %d\t size: %d\n", (int)sbuf.st_ino, (int)sbuf.st_size);

    int i = 1;

    while(i < strlen(c))
    {
        fprintf(fd1, "%c", c[i]);
        i += 2;
    }

    if (pthread_join(tid, NULL))
    {
        printf("ERROR: in joining thread");
        return -1;
    }
    if (err)
    {
        printf("ERROR: in reading in thread");
        return -1;
    }
}

```

```

fclose(fd1);
stat(FILENAME, &sbuf);
printf("\n>>> inode: %d\t size: %d\n", (int)sbuf.st_ino, (int)sbuf.st_size);
printf("\n");
return 0;
}

```

Результат работы

```

>>> inode: 1313305      size: 0
>>> inode: 1313305      size: 0
>>> inode: 1313305      size: 15
>>> inode: 1313305      size: 15

```

bdfhjlnprtvxzBC

Анализ работы

Многопоточная версия программы. В файле «alphabet_3.txt» также останется информация из того дескриптора, для которого flose() был вызван последним.

Схема

