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»

Лабораторная работа № 5
Тема: функций ввода-вывода в UNIX/Linux
Дисциплина: Операционные системы

Студент Лучина Е.Д

Группа ИУ7-61Б

Преподаватель Рязанова Н.Ю.

Москва.
2020 г.

Отчет должен:

- содержать коды программ;
- скриншоты результата выполнения каждой программы;
- анализ полученного результата;
- рисунок, демонстрирующий созданные дескрипторы и связь между ними.
- структуру FILE

Первая программа (testCIO.c)

ex01.c

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

int main() {
    /* have kernel open connection to file alphabet.txt*/
    int fd = open("alphabet.txt", O_RDONLY);

    /* create two a C I/O buffered streams using the above connection*/
    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);

    /*read a char & write it alternatingly from fs1 and fs2*/
    int flag1 = 1, flag2 = 2;
    char c;
    while (flag1 == 1 || flag2 == 1)
    {
        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;
}
```

```
lena@lena-Aspire-One-522:~/myOSlabs/lab05$ gcc ex01.c
lena@lena-Aspire-One-522:~/myOSlabs/lab05$ ls
alphabet.txt  a.out  ex01.c
lena@lena-Aspire-One-522:~/myOSlabs/lab05$ cat -e alphabet.txt
abcdefghijklmnopqrstuvwxyz$
lena@lena-Aspire-One-522:~/myOSlabs/lab05$ ./a.out
aubvcwdxeyfzg
hijklmnopqrstlena@lena-Aspire-One-522:~/myOSlabs/lab05$
```

Системный вызов open() открывает файл, имя которого указанно в качестве первого аргумента. Второй аргумент - режим доступа. O_RDONLY (open read only) означает открыть только для чтения, и в случае отсутствия файла или отсутствии прав open вернет -1. При успешном выполнении вывоз возвращает дескриптор файла - небольшое неотрицательное целое число, которое используется для ссылки на

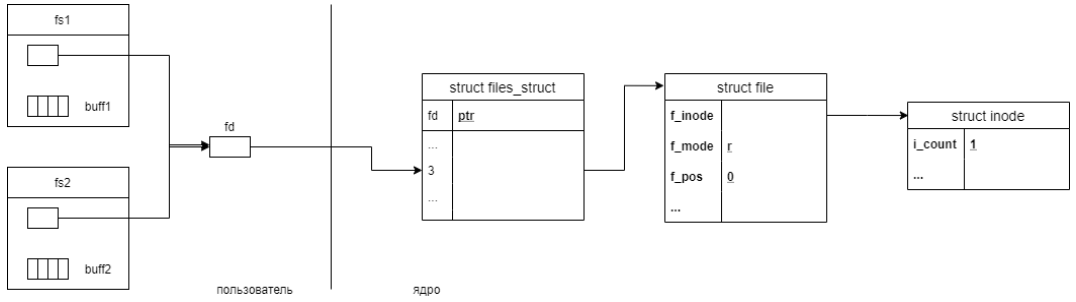
открытый файл. Смещение файла устанавливается на начало файла. Вызов open() создает новый дескриптор открытого файла – struct file в общесистемной таблице открытых файлов.

Далее с помощью функции fdopen(), по дескриптору файла fd, с открытым файлом связывается поток данных для чтения (режим доступа равен “r”).Режим доступа в fdopen() должен совпадать с режимом, в котором открыт файл. Откроем два потока.

Создается два статических буфера размером 20 байт, которые с помощью функций setvbuf связываются с файлом. Режим _IOFBF означает полную буферизацию, то есть данные будут буферизироваться, пока буфер не заполниться полностью.

После чего в цикле поочередно считывается один символ из файла fscanf(fs, "%c", &c) и выводится в стандартный поток вывода - stdout fprintf(stdout, "%c", c). Однако, при чтении из файла происходит буферизация. При первом считывании из потока fs1 буфер buff1 заполнится первыми 20 символами содержимого файла - abcdefghijklmnopqrst, а при следующем вызове fscanf(fs2,"%c", &c) второй буфер buff2 заполнится оставшимися символами – uvwxyz. Обе структуры FILE ссылаются на один и тот же дескриптор файла, и при первом чтении указатель в файле смещается на максимальные для первого буфера 20 байт.

Таким образом, посимвольные чтение и вывод фактически происходят не из файла, а из буферов, что и дает такой результат.



Вторая программа (testKernelIO.c)

ex02.c

```
#include <unistd.h>
#include <fcntl.h>
int main()
{
        char c;

        /* have kernel open two connection to file alphabet.txt */
        int fd1 = open("alphabet.txt", O_RDONLY);
        int fd2 = open("alphabet.txt", O_RDONLY);

        int flag = 1;
        while(flag)
        {
                if (read(fd1, &c, 1) == 1) write(1, &c, 1);
```

```
                else flag = 0;

                if (read(fd2, &c, 1) == 1) write(1, &c, 1);
                else flag = 0;
        }
        return 0;
}
```

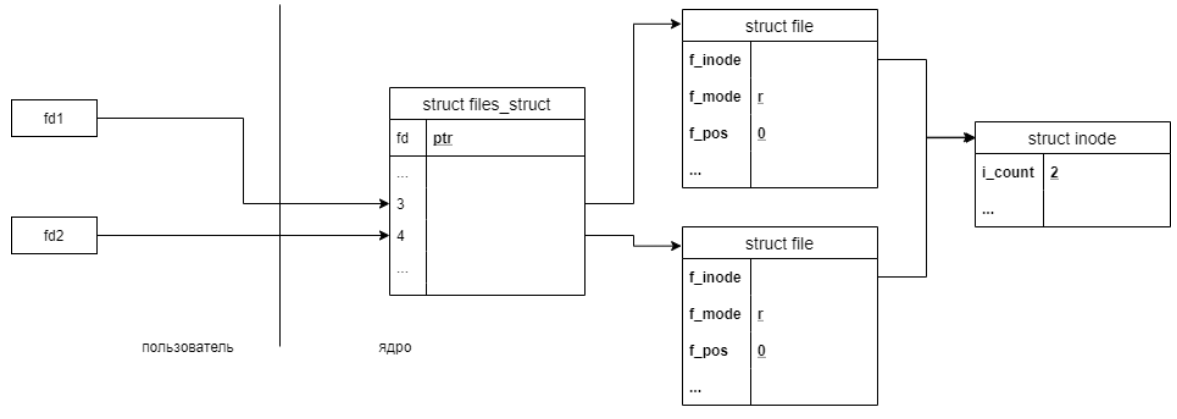
```
lena@lena-Aspire-One-522:~/myOSlabs/lab05$ gcc ex02.c
lena@lena-Aspire-One-522:~/myOSlabs/lab05$ ls
alphabet.txt  a.out  ex01.c  ex02.c
lena@lena-Aspire-One-522:~/myOSlabs/lab05$ cat -e alphabet.txt
abcdefghijklmnopqrstuvwxyz$
lena@lena-Aspire-One-522:~/myOSlabs/lab05$ ./a.out
aabbccddeeffgghhiijjkkllmmnnooppqqrrssttuuvvwwxxyyzz

lena@lena-Aspire-One-522:~/myOSlabs/lab05$
```

Здесь два вызова open(), каждый из которых создает новый дескриптор файла в системной таблице файлов и новую запись в системной таблице открытых файлов. Записи являются независимыми и каждая имеет свой указатель на позицию в файле.

В цикле с помощью функций read() и write() происходит посимвольное чтение из файла. read(int fd, void *buf, unsigned len) делает попытку считать len байт из файла, связанного с fd, в буфер, адресуемый параметром buf. При успешном считывании возвращает количество считанных байт. write(int fd, void *buf, int len) записывает len байт из буфера buf в файл, связанный с fd. В данном случае считываем и выводим один символ за итерацию, вывод происходит в стандартный поток вывода - stdout - 1, в качестве буфера выступает переменная c.

Таким образом происходит последовательное считывание и вывод каждого символа по обоим дескрипторам.



Другой вариант программы с вызовом функции open().

ex02_2.c

```
#include <unistd.h>
#include <fcntl.h>
int main()
```

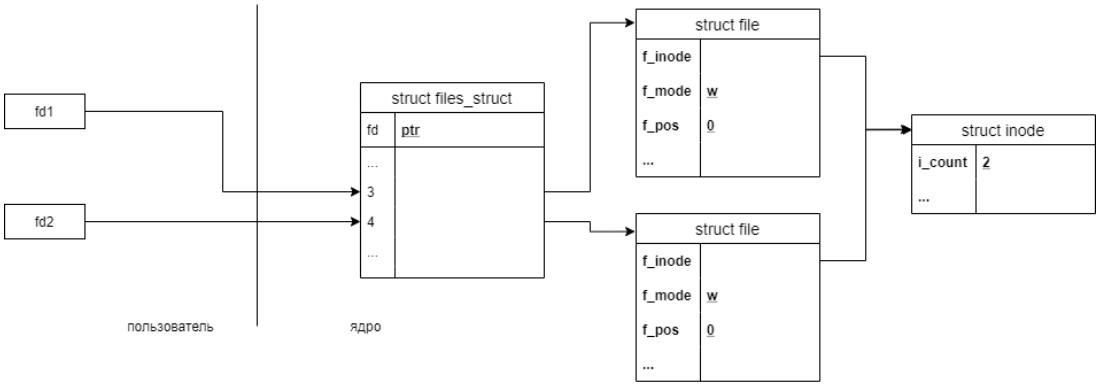
```c
{
        int fd1 = open("q.txt", O_RDWR);
        int fd2 = open("q.txt", O_RDWR);
        int curr = 0;
        for(char c = 'a'; c <= 'z'; c++)
        {
                if (c%2)
                        write(fd1, &c, 1);
                else
                        write(fd2, &c, 1);
        }
        close(fd1);
        close(fd2);
        return 0;
}
```

```
lena@lena-Aspire-One-522:~/myOSlabs/lab05$ gcc ex02_2.c
lena@lena-Aspire-One-522:~/myOSlabs/lab05$ touch q.txt
lena@lena-Aspire-One-522:~/myOSlabs/lab05$ ls
alphabet.txt  a.out  ex01.c  ex02_2.c  ex02.c  q.txt
lena@lena-Aspire-One-522:~/myOSlabs/lab05$ cat -e q.txt
lena@lena-Aspire-One-522:~/myOSlabs/lab05$ ./a.out
lena@lena-Aspire-One-522:~/myOSlabs/lab05$ cat -e q.txt
bdfhjlnprtvxzlena@lena-Aspire-One-522:~/myOSlabs/lab05$
```

Так же, как в программе выше, два вызова open() создают два новых дескриптора файла “q.txt” в системной таблице файлов и две новые записи в системной таблице открытых файлов. Записи являются независимыми и каждая имеет свой указатель на позицию в файле. Файл открывается в режиме O_RDWR (read, write), позволяющей читать и записывать в файл.

В цикле при записи символов в файл происходит перезапись содержимого. Сначала вывоз `write(fd1, &c, 1);` запишет в начало файла ‘a’, потом второй вывоз `write(fd2, &c, 1);` запишет ‘b’ на то же место - в начало. В обоих случаях после записи позиция перемещается дальше. В результате в файле имеем четные символы.

После записи файлы закрываются с помощью вызова close().



Третья программа

Написать программу, которая открывает один и тот же файл два раза с использованием библиотечной функции fopen(). Для этого объявляются два файловых дескриптора. В цикле записать в файл буквы латинского алфавита поочередно передавая функции fprintf() то первый дескриптор, то – второй.
ex03.c

```
#include <stdio.h>
#include <fcntl.h>
int main()
{
	FILE *fs1 = fopen("a.txt", "w");
	FILE *fs2 = fopen("a.txt", "w");
	for(char c = 'a'; c <= 'z'; c++)
	{
		if (c % 2)
			fprintf(fs1, "%c", c);
		else
			fprintf(fs2, "%c", c);
	}
	fclose(fs1);
	fclose(fs2);
	return 0;
}
```

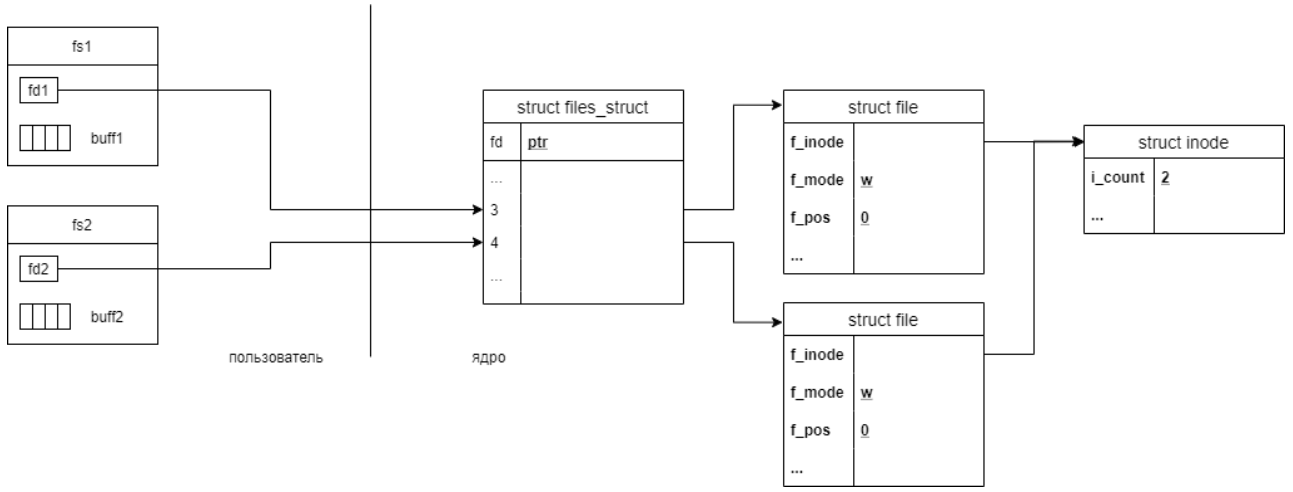
```
lena@lena-Aspire-One-522:~/myOSlabs/lab05$ gcc ex03.c
lena@lena-Aspire-One-522:~/myOSlabs/lab05$ ls
alphabet.txt  a.out  ex01.c  ex02_2.c  ex02.c  ex03.c  q.txt
lena@lena-Aspire-One-522:~/myOSlabs/lab05$ ./a.out
lena@lena-Aspire-One-522:~/myOSlabs/lab05$ ls
alphabet.txt  a.out  a.txt  ex01.c  ex02_2.c  ex02.c  ex03.c  q.txt
lena@lena-Aspire-One-522:~/myOSlabs/lab05$ cat -e a.txt
bdfhjlnprtvxzlena@lena-Aspire-One-522:~/myOSlabs/lab05$
```

С помощью двух вызовов fopen("a.txt", "w") создаются два дескриптора файла “a.txt” в таблице файлов, открытых процессом, две записи в системной таблице открытых файлов и два потока fs1 и fs2, которые связываются с этими дескрипторами. Отметим, что файл открыт на запись - режим доступа “w”.

Функция fprintf() обеспечивает буферизацию, таким образом запись производится в два разных буфера поочередно: в первом буфере оказываются нечетные символы, во втором четные.

Запись в сам файл осуществляется только в случае переполнения буфера, вызове функции fflush() или вызове fclose(), внутри которой осуществляется вызов fflush(), если поток использовался для вывода данных.

Таким образом, при первом вызове fclose() осуществляется запись в файл данных из первого буфера. Затем при втором вызове fclose() осуществляется перезапись данных файла из второго буфера. В результате имеет файл с четными символами алфавита.



Структура 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;
    #if 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
};
```