

с



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

По предмету: «Операционные системы»

Тема: Буферизованный и не буферизованный
ВВОД-ВЫВОД

Студент: Лаврова А. А.,
Группа: ИУ7-65Б
Преподаватель: Рязанова Н. Ю.

Москва, 2020 г.

№1

Листинг программы:

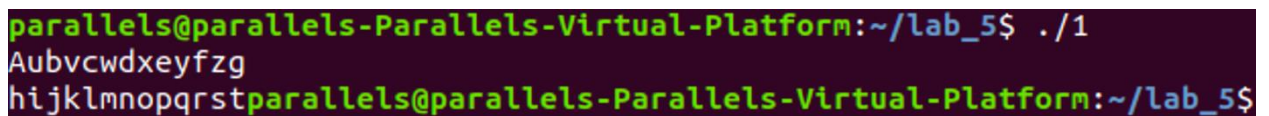
```
#include <stdio.h>
#include <fcntl.h>

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:



```
parallels@parallels-Parallels-Virtual-Platform:~/lab_5$ ./1
Aubvcwdxeyfzg
hijklmnopqrstparallels@parallels-Parallels-Virtual-Platform:~/lab_5$
```

Системный вызов `open()` создает дескриптор файла. Так как передается флаг `O_RDONLY`, файл открывается только на чтение, при этом указатель устанавливается на начало файла. Внутри системного вызова вызываются другие функции, в том числе `get_unused_fd()`, которая ищет пустой слот в таблице дескрипторов файлов процесса. Если пустого дескриптора нет, то возвращается ошибка, в ином случае возвращается наименьший файловый

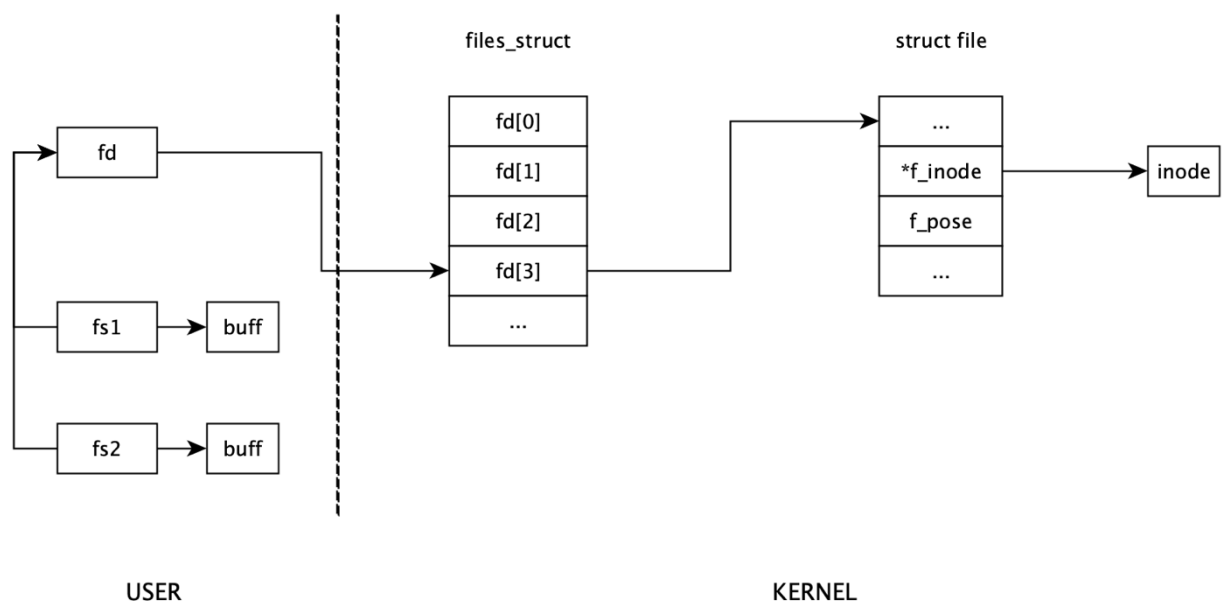
дескриптор из этой таблицы. Также вызывается функция `filp_open()`, заполняющая структуру `struct file` для данного открытого файла.

После создания дескриптора функция `fdopen()` создает два объекта типа `FILE` и связывает два потока для чтения с данным дескриптором.

Функция `setvbuf()` изменяет тип буферизации на блочную размеров в 20 байт.

В цикле при первом вызове `fscanf()` первый буфер файловой структуры заполняется 20 символами от «A» до «t», так как размер буфера установлен как 20 байт. Функция `fscanf()` напечатает первый символ из буфера файловой структуры `fs1` – «A». Так как оба объекта связаны одним и тем же дескриптором, то позиция в файле для них будет определяться одним полем `f_pos` структуры `struct file`. Из-за этого при втором вызове `fscanf()` во другом буфере файловой структуры будут оставшиеся 6 символов от «u» до «z». Функция `fscanf()` напечатает первый символ из буфера файловой структуры `fs2` – «u». Далее в результате поочередного вывода символов из каждого буфера потока `stdout` получим результат «Aubvcwdxeyfzghijklmnopqrst».

Связь структур:



№2

Листинг программы:

```
#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)
    {
        if (read(fd1, &c, 1) != 1)
            flag = 0;
        write(1, &c, 1);

        if (read(fd2, &c, 1) != 1)
            flag = 0;
        write(1, &c, 1);
    }

    return 0;
}
```

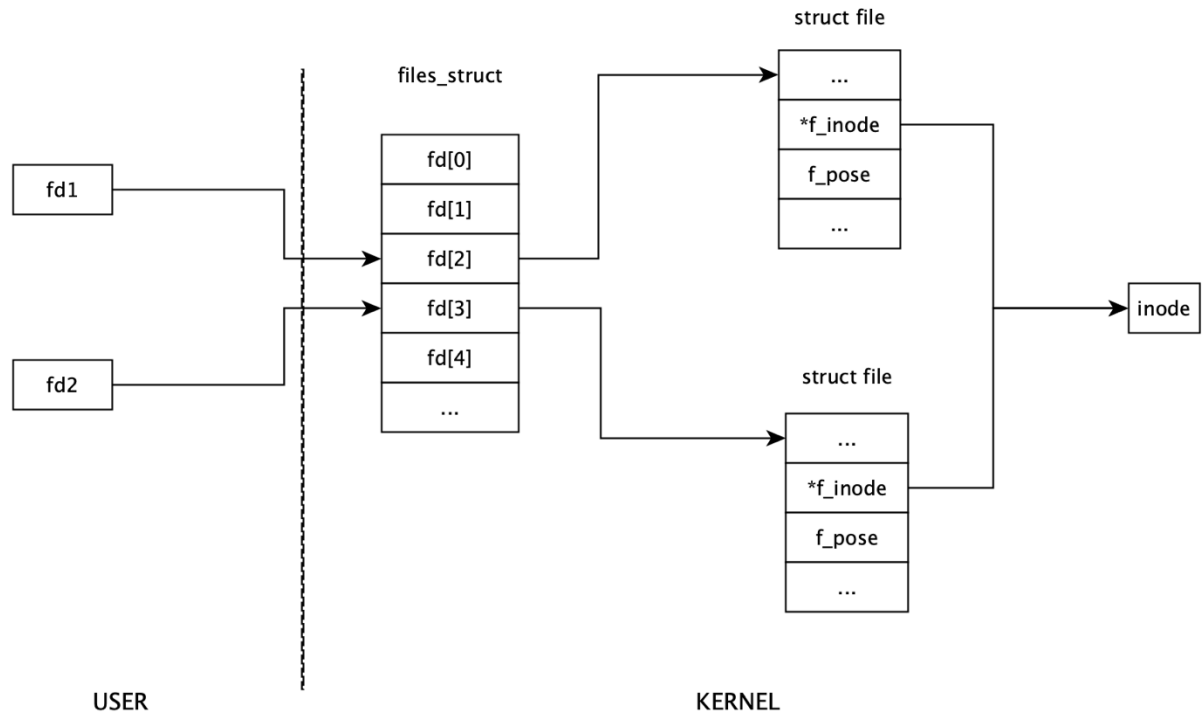
Результат работы программы:

```
parallels@parallels-Parallels-Virtual-Platform:~/lab_5$ ./2
AAbbccddeeffgghhiijjkkllmmnnooppqrrssttuuvvwwxxyyzz
```

С помощью системного вызова `open()` создается два файловых дескриптора одного и того же файла, открытого на чтение (флаг `O_RDONLY`).

Следовательно, создаются две различные структуры `struct file`, и положения указателей в файле не зависят друг от друга. Поэтому в ходе выполнения цикла, с использованием системных вызовов `read()` и `write()`, из файла считывается и записывается в стандартный поток один и тот же символ два раза.

Связь структур:



№3

Листинг программы:

```
#include <stdio.h>

int main()
{
    FILE* fd[2];
    fd[0] = fopen("alphabet.txt", "w");
    fd[1] = fopen("alphabet.txt", "w");

    char* abc = "abcdefghijklmnopqrstuvwxyz";

    for (i = 0; i < 26; ++i)
    {
        if (i % 2 == 0)
            fprintf(fd[0], "%c", abc[i]);
        else
            fprintf(fd[1], "%c", abc[i]);
    }

    fclose(fd[0]);
    fclose(fd[1]);
    return 0;
}
```

Результат работы программы:

bdfhjlnprtvxz

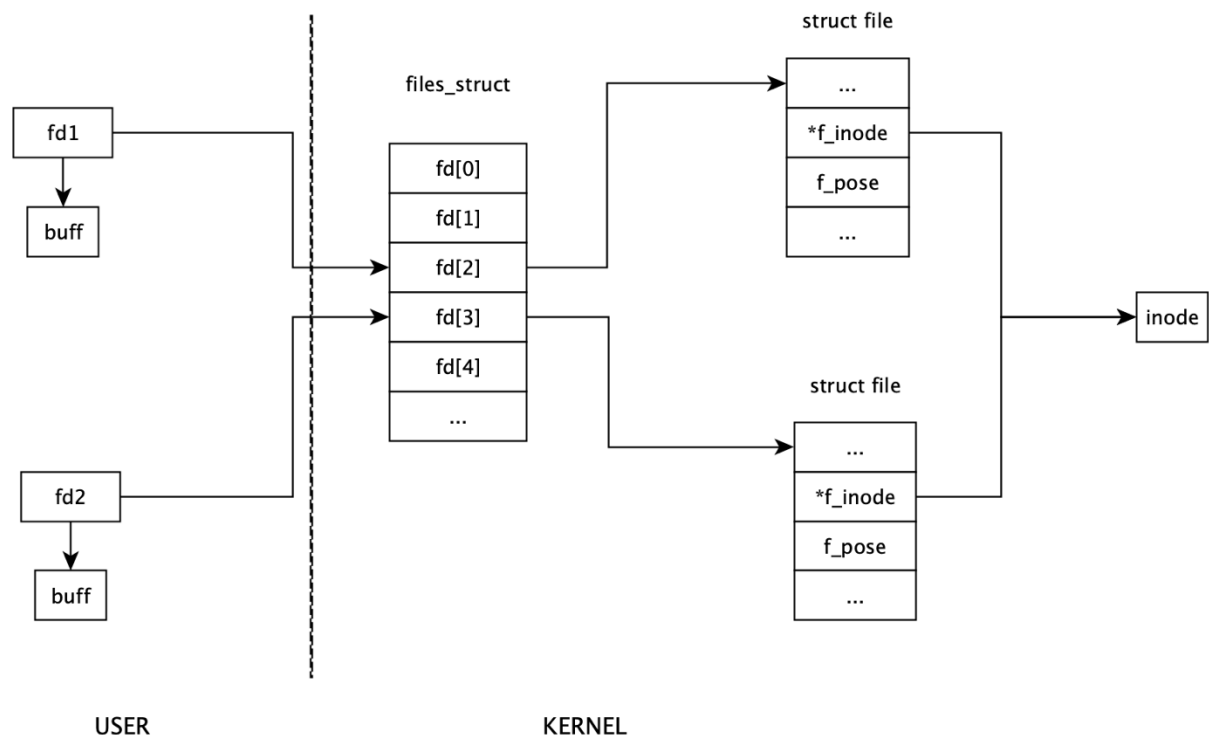
Функция `fork()` является стандартной функцией библиотеки `stdio.h`, которая выполняет ввод/вывод с буферизацией. С ее помощью создаем два независимых потока для ввода. Каждый из них имеет свой файловый дескриптор, и положение указателей в файле независимы.

Нечетные буквы алфавита (если нумерация ведется с 1, или четные, если нумерация ведется с 0) записываются в первый буфер, а четные – во второй буфер.

Однако запись непосредственно в файл происходит при вызове `fclose()` и `fflush()`. Сначала вызывается `fclose(fd[0])` и в файл записываются все нечетные буквы алфавита («асегіктоqсуу»), которые содержались в буфере файлового дескриптора `fd[0]`. Но так как у объектов `FILE` разные структуры `struct file`, то и значения полей `f_pos` независимы друг от друга.

Следовательно, запись второго буфера после вызова `fclose(fd[1])` будет выполнена с начала файла, и данные в `alphabet.txt` будут перезаписаны.

Связь структур:



Листинг структуры FILE:

```
struct FILE {  
    ssize_t _cnt; // число байт в буфере _cnt  
    unsigned char *_ptr; // указатель на следующий  
    //символ, который подлежит чтению или записи _ptr  
    unsigned char *_base; // указатель на буфер _base  
    unsigned char _flag; // флаги состояния потока _flag  
    unsigned char _file; // указатель на файловый дескриптор _file, с которым  
    ассоциирован данный поток  
    ...  
};
```

Вывод

Исходя из вышеперечисленных примеров можно сделать выводы:

- 1) При небуферизированном вводе/выводе, что создается столько дескрипторов открытого файла, сколько раз был открыт файл (важно при одновременном открытии одного и того же файла). Каждый дескриптор имеет поле `f_rpos`, которое указывает на позицию чтения или записи в логическом файле.
- 2) При буферизированном вводе/выводе необходимо учитывать факт записи/чтения данных из буфера, так как неосторожные действия с данными могут привести к неправильной последовательности данных или к даже к их потере.