|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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»*

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 05 |

**Дисциплина:  *Операционные системы***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ***ИУ7И-66Б*** |  |  | **Нгуен Ф. С.** |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | **Рязанова Н. Ю.** |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

*Москва, 2021*

**Структура \_IO\_FILE** (/usr/include/x86\_64-linux-gnu/bits/types/struct\_FILE.h)

struct \_IO\_FILE

{

int \_flags; /\* High-order word is \_IO\_MAGIC; rest is flags. \*/

/\* The following pointers correspond to the C++ streambuf protocol. \*/

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;

\_\_off\_t \_old\_offset; /\* This used to be \_offset but it's too small. \*/

/\* 1+column number of pbase(); 0 is unknown. \*/

unsigned short \_cur\_column;

signed char \_vtable\_offset;

char \_shortbuf[1];

\_IO\_lock\_t \*\_lock;

#ifdef \_IO\_USE\_OLD\_IO\_FILE

};

1. **Первая программа**

#include <stdio.h>

#include <fcntl.h>

#define OK 0

#define BUF\_SIZE 20

#define FILE\_NAME "./atoz.txt"

int main(void)

{

int fd = open(FILE\_NAME, O\_RDONLY);

FILE \*fs1 = fdopen(fd, "r");

char buff1[BUF\_SIZE];

setvbuf(fs1, buff1, \_IOFBF, BUF\_SIZE);

FILE \*fs2 = fdopen(fd, "r");

char buff2[BUF\_SIZE];

setvbuf(fs2, buff2, \_IOFBF, BUF\_SIZE);

int flag1 = 1, flag2 = 2;

while (flag1 == 1 || flag2 == 1)

{

char c;

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

{

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

}

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

{

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

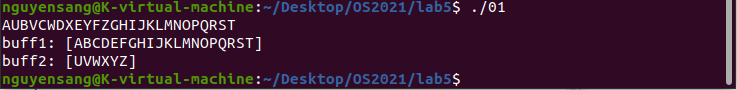
}

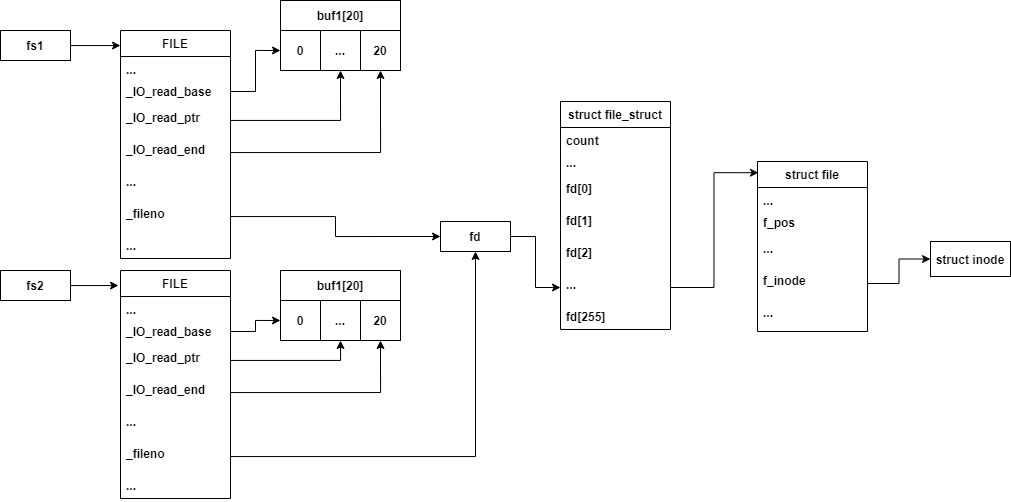
}

fprintf(stdout, "\nbuff1: [%s]\nbuff2: [%s]\n", buff1, buff2);

return OK;

}





**Анализ работы**

1. Функция **open()** создает новый файловый дескриптор файла "atoz.txt", запись в системной таблице открыт файлов. Эта запись регистрирует смещение в файле и флаги состояния файла;
2. функция **fdopen()** создает указатели на структуру **FILE**. Поле \_fileno содержит дескриптор, который вернула функция **open**();
3. функция **setvbuf**() задает размер буффера в 20 байт и меняет тип буферизации на полную;
4. при первом вызове функции **fscanf**() в цикле (для **fs1**), **buff1** будет заполнен полностью -- первыми 20 буквами алфавита. **f\_pos** в структуре **struct\_file** открытого файла увеличится на 20;
5. при втором вызове **fscanf**() в цикле (для **fs2**) буффер **buff2** будет заполнен оставшимися 6 символами (начиная с **f\_pos**);
6. в цикле выводятся символы из **buff1** и **buff2**.
7. **Вторая программа**

#include <fcntl.h>

#include <unistd.h>

#define OK 0

#define FILE\_NAME "./atoz.txt"

int main(void)

{

int fd1 = open(FILE\_NAME, O\_RDONLY);

int fd2 = open(FILE\_NAME, O\_RDONLY);

int rc1, rc2 = 1;

while (rc1 == 1 || rc2 == 1)

{

char c;

rc1 = read(fd1, &c, 1);

if (rc1 == 1)

{

write(1, &c, 1);

}

rc2 = read(fd2, &c, 1);

if (rc2 == 1)

{

write(1, &c, 1);

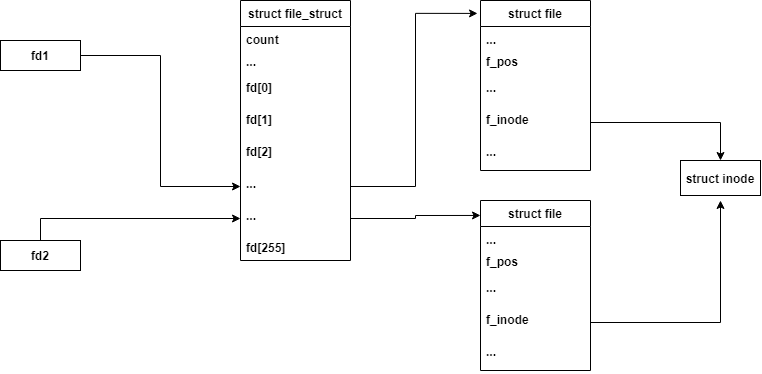
}

}

return OK;

}





**Анализ работы**

1. Функция **open**() создает файловые дескрипторы, два раза для одного и того же файла, поэтому в программе существует две различные **struct** **file**, но ссылающиеся на один и тот же **struct** **inode**;
2. Из-за того что структуры разные, посимвольная печать просто дважды выведет содержимое файла в формате ‘aabbcc...’;
3. **Третья программа**

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#define OK 0

#define FILE\_NAME "./out.txt"

int main()

{

FILE \*f1 = fopen(FILE\_NAME, "w");

FILE \*f2 = fopen(FILE\_NAME, "w");

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

{

if (c % 2)

{

fprintf(f1, "%c", c);

}

else

{

fprintf(f2, "%c", c);

}

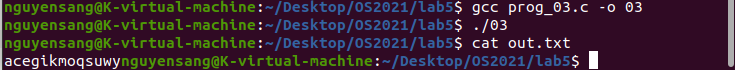
}

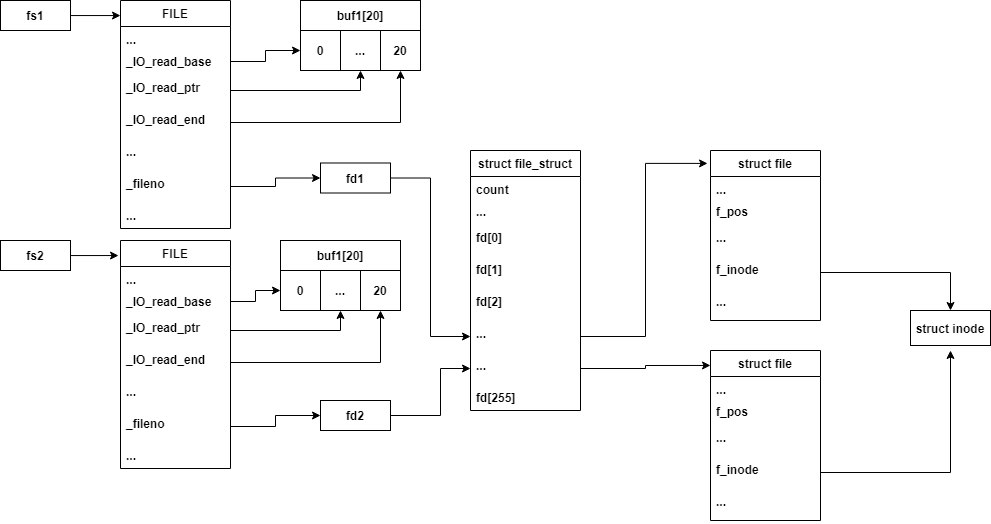
fclose(f2);

fclose(f1);

return OK;

}





**Анализ работы**

1. Файл открывается на запись два раза, с помощью функции fopen(). Для этого объявляются два файловых дескриптора.;
2. По умолчанию используется полная буфферизация, при которой запись в файл из буфера произойдет либо при заполнении буфера, либо при вызове fclose(), либо при завершении процесса;
3. В цикле записываются в файл буквы латинского алфавита поочередно передавая функции fprintf() то первый дескриптор, то – второй;
4. Из-за того f\_pos независимы для каждого дескриптора файла, запись в файл будет производится с самого начала;
5. Таким образом, информация записаная при первом вызове fclose() будет потеряна в результате второго вызова fclose();