|  |  |
| --- | --- |
|  | **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**

*к лабораторной работе №5*

*По курсу: «Операционные системы»*

*На тему: «Буферизованный и не буферизованный*

*ввод-вывод»*

Студент ИУ7-62Б

Блохин Д.М.

Преподаватель

Рязанова Н.Ю.

*Москва, 2021 г.*

Оглавление

[Структура \_IO\_FILE(/usr/include/bits/types/struct\_FILE.h): 3](#_Toc38235502)

[Первая программа 4](#_Toc38235503)

[Листинг первой программы: 4](#_Toc38235504)

[Результаты работы первой программы: 5](#_Toc38235505)

[Связь структур: 5](#_Toc38235506)

[Анализ работы первой программы: 5](#_Toc38235507)

[Вторая программа 6](#_Toc38235508)

[Листинг второй программы: 6](#_Toc38235509)

[Результат работы второй программы: 7](#_Toc38235510)

[Связь структур: 7](#_Toc38235511)

[Анализ работы второй программы: 7](#_Toc38235512)

[Третья программа 8](#_Toc38235513)

[Листинг третьей программы: 8](#_Toc38235514)

[Результат работы третьей программы: 8](#_Toc38235515)

[Связь структур: 9](#_Toc38235516)

[Анализ работы третьей программы: 9](#_Toc38235517)

**Задача:** анализ особенностей работы функций ввода-вывода в UNIX/Linux.

В лабораторной работе анализируется результат выполнения трех программ. Программы демонстрируют открытие одного и того же файла несколько раз. Необходимо проанализировать структуру FILE, работу программ и объяснить результаты их работы, привести рисунок, демонстрирующий созданные дескрипторы и связь между ними.

**Файл /usr/include/bits/types/FILE.h:**

typedef struct \_IO\_FILE FILE;

## Структура \_IO\_FILE(/usr/include/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; // индекс в массиве fd структуры files\_struct

int \_flags2;

...

\_IO\_lock\_t \*\_lock;

};

**Структура struct:**

|  |
| --- |
| **struct** **stat** {     **dev\_t** st\_dev; */\* устройство \*/*     **ino\_t** st\_ino; */\* inode \*/*     **mode\_t** st\_mode; */\* режим доступа \*/*     **nlink\_t** st\_nlink; */\* количество жестких ссылок \*/*     **uid\_t** st\_uid; */\* идентификатор пользователя-владельца \*/*     **gid\_t** st\_gid; */\* идентификатор группы-владельца \*/*     **dev\_t** st\_rdev; */\* тип устройства \*/*  */\* (если это устройство) \*/*     **off\_t** st\_size; */\* общий размер в байтах \*/*     **blksize\_t** st\_blksize; */\* размер блока ввода-вывода \*/* */\* в файловой системе \*/*     **blkcnt\_t** st\_blocks; */\* количество выделенных блоков \*/*     **time\_t** st\_atime; */\* время последнего доступа \*/*     **time\_t** st\_mtime; */\* время последней модификации \*/*     **time\_t** st\_ctime; */\* время последнего изменения \*/* }; |

# Первая программа

## Листинг первой программы:

//testCIO.c

#include <stdio.h>

#include <fcntl.h>

#define FILENAME "alphabet.txt"

int main()

{

// have kernel open connection to file alphabet.txt

int fd = open(FILENAME, O\_RDONLY);

if (fd == -1)

{

printf("Error opening fd %s\n", FILENAME);

return -1;

}

// 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;

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);

}

}

fprintf(stdout, "\n");

fprintf(stdout, "buf1: %s\nbuf2: %s\n", buff1, buff2);

return 0;

}

## Результаты работы первой программы:

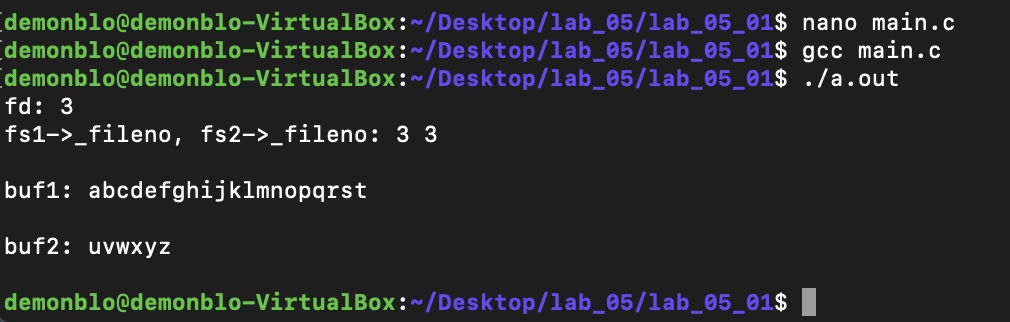


Рис.1 – результат работы первой программы

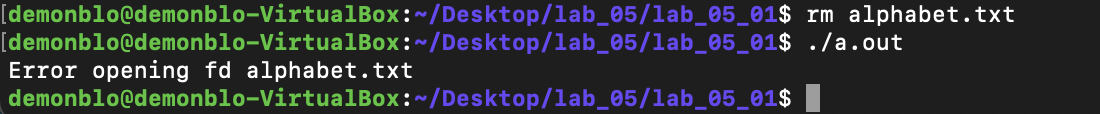


Рис.2 – результат работы, если файла не существует

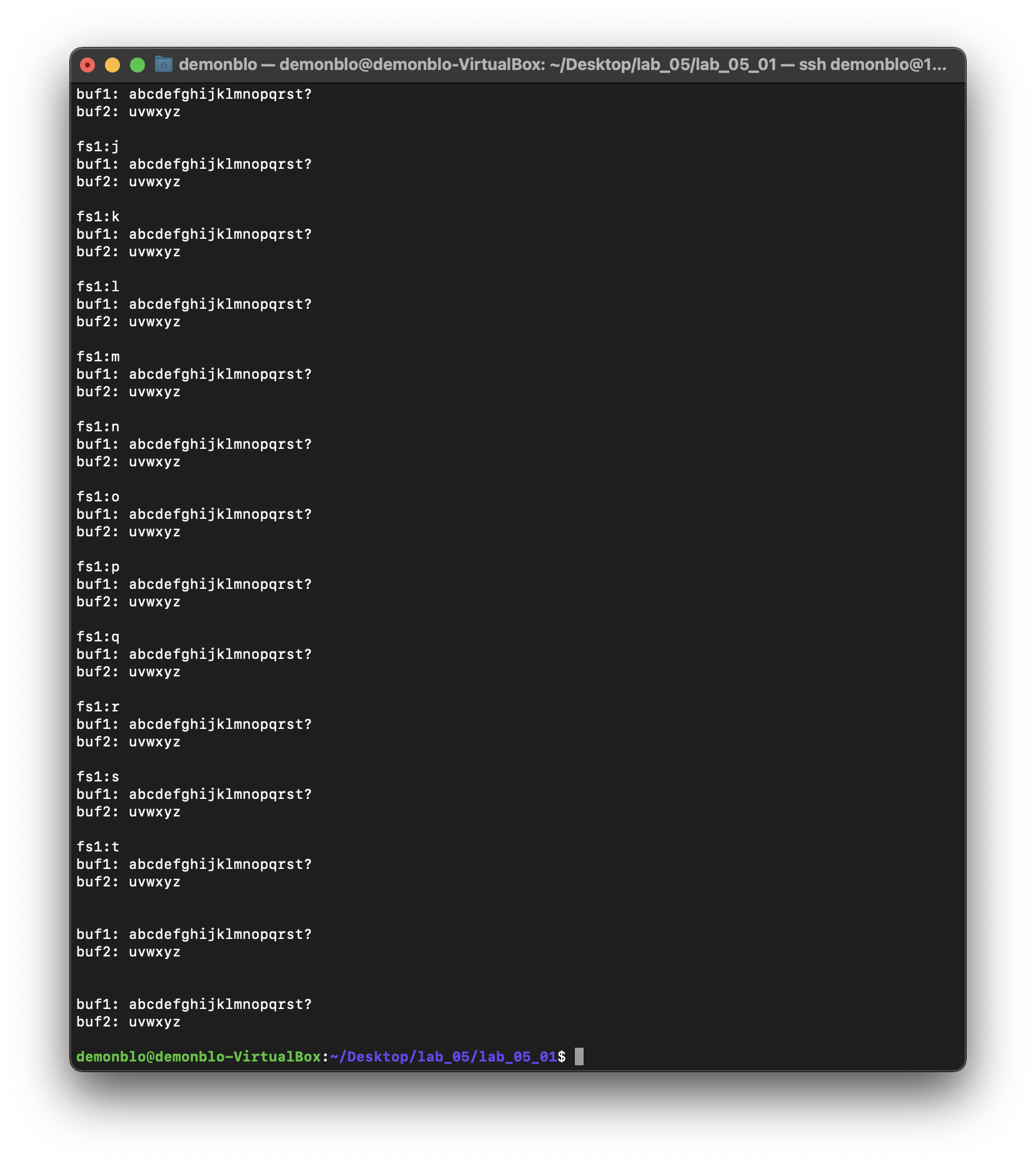
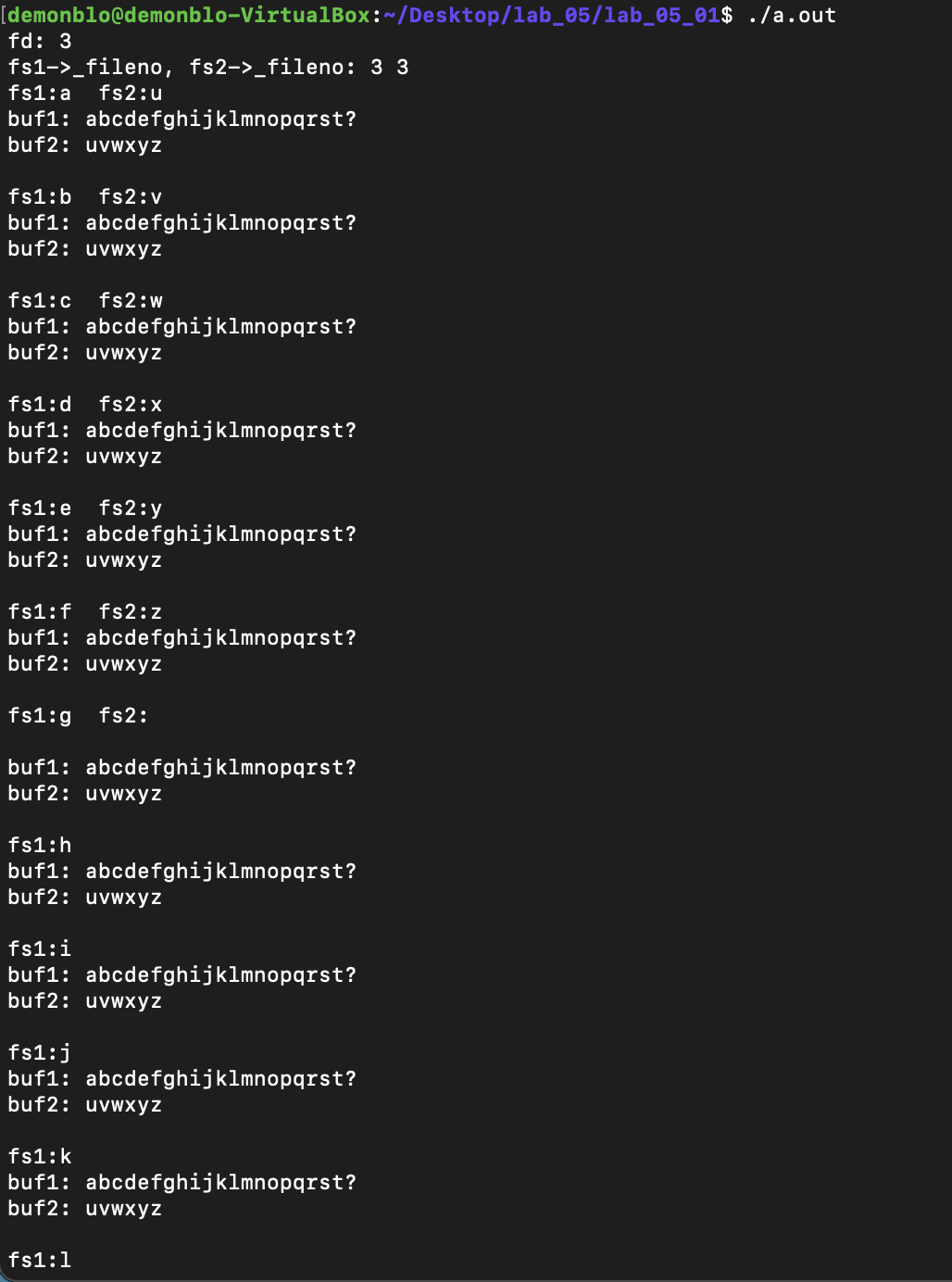


Рис.3 – Полный результат работы программы

**Листинг первой программы для нескольких потоков:**

**#include <unistd.h>**

**#include <stdio.h>**

**#include <fcntl.h>**

**#include <pthread.h>**

**#include <stdlib.h>**

**#define** **FILENAME** **"alphabet.txt"**

**#define** **BUFF\_SIZE** 20

**#define** **COUNT\_THREADS** 2

void\* thread\_func(void \*fs) {

char c;

char buff2[**BUFF\_SIZE**];

setvbuf(fs, buff2, **\_IOFBF**, **BUFF\_SIZE**);

**while** (fscanf((**FILE** \*) fs, **"%c"**, &c) == 1) {

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

}

fprintf(stdout,**"\nbuf2: %s"**, buff2);

}

int main() {

int fd = open(**FILENAME**, **O\_RDONLY**);

**if** (fd == -1) {

printf(**"Error opening fd %s\n"**, **FILENAME**);

return -1;

}

printf(**"fd: %d\n\n"**, fd);

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

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

printf(**"fs1->\_fileno is %d,\n fs2->\_fileno is %d\n"**, fs1->\_fileno, fs2->\_fileno);

char buff1[**BUFF\_SIZE**];

setvbuf(fs1, buff1, **\_IOFBF**, **BUFF\_SIZE**);

pthread\_t tid;

pthread\_create(&tid, **NULL**, thread\_func, (void \*) fs2);

char c;

**while**(fscanf(fs1, **"%c"**, &c) == 1) {

sleep(1);

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

}

void \*res;

pthread\_join(tid, &res);

fprintf(stdout, **"\n"**);

fprintf(stdout, **"buf1: %s\n"**, buff1);

return 0;

}

Результат работы:

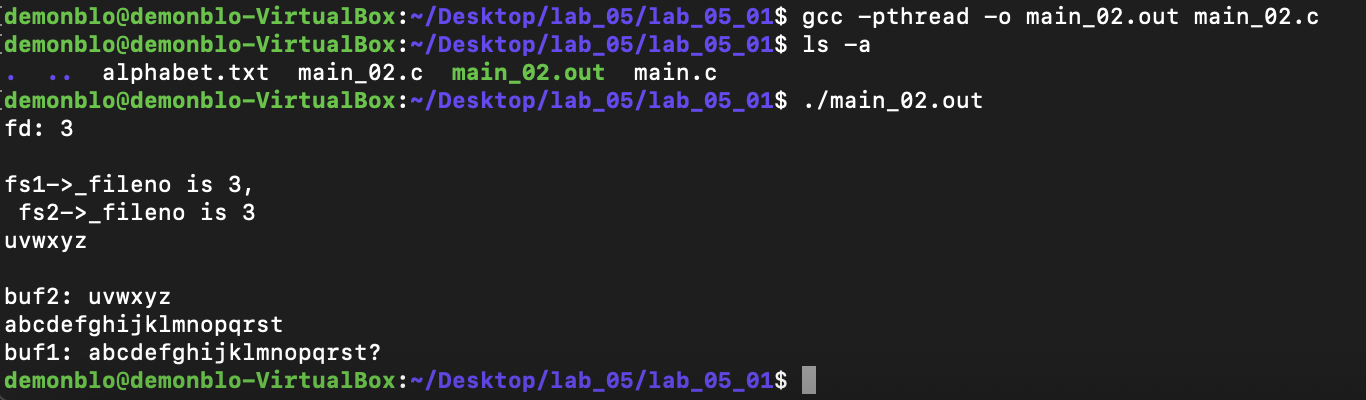


Рис. 4 - результат работы первой программы для нескольких потоков

## Связь структур:

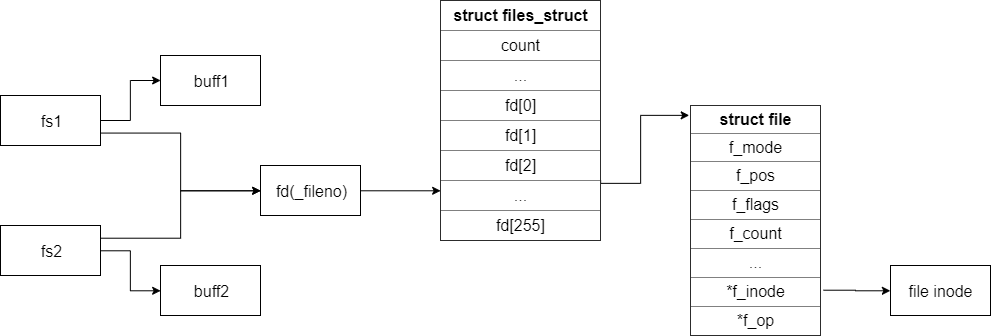


Рис.5 - связь структур первой программы

## Анализ работы первой программы:

С помощью системного вызова open() создается дескриптор открытого на чтение файла. Системный вызов open() возвращает индекс в массиве fd структуры files\_struct.

Далее с помощью функции fdopen() создается fs1 и fs2 - структуры типа FILE, связанные с fd. Создаются два буффера buff1 и buff2 размером 20 байт, с помощью функции setvbuf() для fs1 и fs2 задаются соответствующие буфферы и изменяется тип бефферизации на полную.

Далее в цикле происходит поочередный вызов fscanf() для fs1 и fs2. Так как установлена полная буфферизация, то при вызове fscanf() буфер будет заполнен полностью либо вплоть до конца файла, а f\_pos установится на следующий за последним записанным в буфер символ. Также внутри цикла будут поочередно выводится символы из buff1 и buff2 до тех пор, пока символы в одном из буфферов не закончатся. Тогда на экран будут последовательно выведены оставшиеся символы из другого буфера.

# Вторая программа

## Листинг второй программы:

#include <fcntl.h>

#include <unistd.h>

int main()

{

char c;

int flag = 1;

// have kernel open two connection to file alphabet.txt

int fd1 = open("alphabet.txt", O\_RDONLY);

int fd2 = open("alphabet.txt", O\_RDONLY);

// read a char & write it alternatingly from connections fs1 & fd2

while(flag)

{

if (read(fd1, &c, 1) == 1)

{

write(1, &c,1);

if (read(fd2,&c,1) == 1)

write(1, &c, 1);

else

flag = 0;

}

else

{

flag = 0;

}

}

return 0;

}

## Результат работы второй программы:

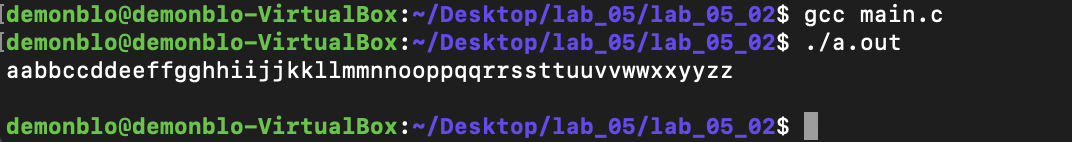
****

Рис. 6 - результат работы второй программы

**Листинг второй программы для нескольких потоков:**

**#include <stdio.h>**

**#include <fcntl.h>**

**#include <pthread.h>**

**#include <stdlib.h>**

void \*thread\_func(void \*arg) {

**FILE**\* fd = (**FILE**\*)arg;

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

{

fprintf(fd, **"%c"**, c);

}

pthread\_exit(**NULL**);

}

int main()

{

int result;

pthread\_t thread1;

pthread\_t thread2;

**FILE**\* fd[2];ы

**// have kernel open connection to file alphabet.txt**

fd[0] = fopen(**"output\_paral.txt"**,**"w"**);

fd[1] = fopen(**"output\_paral.txt"**,**"w"**);

result = pthread\_create(&thread1, **NULL**, thread\_func, &fd[0]);

**if** (result != 0) {

fprintf(stderr, **"can't create thread"**);

return **EXIT\_FAILURE**;

}

result = pthread\_create(&thread2, **NULL**, thread\_func, &fd[1]);

**if** (result != 0) {

fprintf(stderr, **"can't create thread"**);

return **EXIT\_FAILURE**;

}

result = pthread\_join(thread1, **NULL**);

**if** (result != 0) {

fprintf(stderr, **"can't join thread"**);

return **EXIT\_FAILURE**;

}

result = pthread\_join(thread2, **NULL**);

**if** (result != 0) {

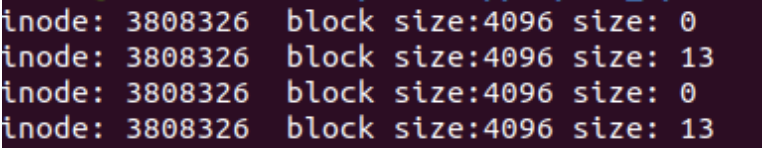
fprintf(stderr, **"can't join thread"**);

return **EXIT\_FAILURE**;

}

return 0;

}

**Вывод программы: **

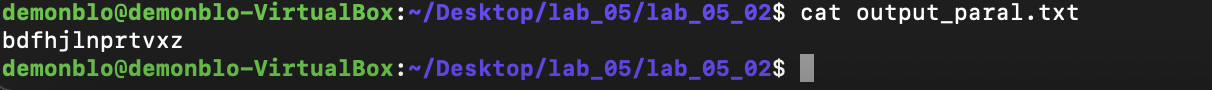
****

Рис. 7-8 - результат работы второй программы для нескольких потоков

## Связь структур:

Рис.9 - связь структур второй программы

## Анализ работы второй программы:

С помощью системного вызова open() создается два дескриптора открытого на чтение файла. Так как дескриптора два, то чтение будет проходить независимо (разные f\_pos) и на экран дважды напечатаются все символы из alphabet.txt.

# Третья программа

## Листинг третьей программы:

#include <stdio.h>

#define FILENAME "os5\_3\_res.txt"

int main()

{

FILE \*fs1 = fopen(FILENAME, "w");

FILE \*fs2 = fopen(FILENAME, "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;

}

## Результат работы третьей программы:

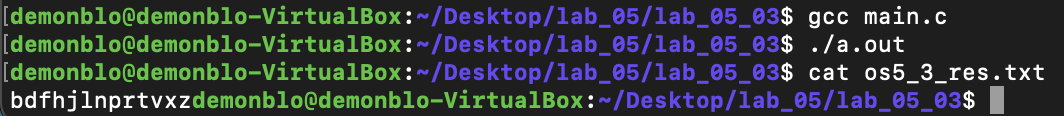
****

Рис. 10 - результат работы третьей программы

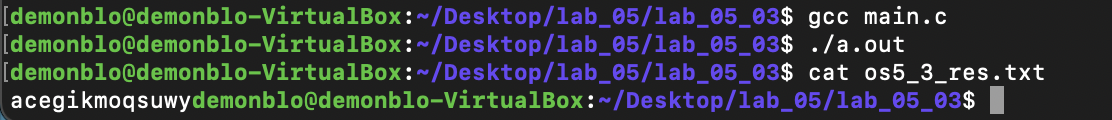
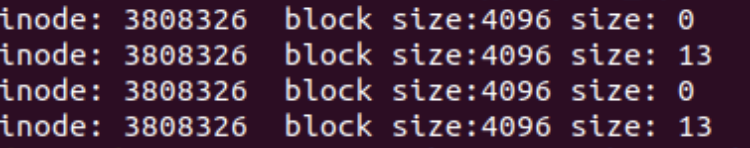


Рис.11 - результат работы третьей программы, если поменять порядок fclose()

**Листинг третьей программы для нескольких потоков:**

**#include <stdio.h>**  
**#include <fcntl.h>**  
**#include <pthread.h>**  
**#include <stdlib.h>**  
  
**void** \***thread\_func**(**void** \*arg) {  
    FILE\* fd = (FILE\*)arg;  
    **for**(**char** c = 'a'; c <= 'z'; c++)  
    {  
        fprintf(fd, "%c", c);  
    }  
    pthread\_exit(NULL);  
}  
**int** **main**()  
{  
    **int** result;  
    **pthread\_t** thread1;  
    **pthread\_t** thread2;  
    FILE\* fd[2];  
    *// have kernel open connection to file alphabet.txt*  
    fd[0] = fopen("output\_paral.txt","w");  
    fd[1] = fopen("output\_paral.txt","w");  
  
    result = pthread\_create(&thread1, NULL, thread\_func, &fd[0]);  
    **if** (result != 0) {  
        fprintf(stderr, "can't create thread");  
        **return** EXIT\_FAILURE;  
    }  
    result = pthread\_create(&thread2, NULL, thread\_func, &fd[1]);  
    **if** (result != 0) {  
        fprintf(stderr, "can't create thread");  
        **return** EXIT\_FAILURE;  
    }  
      
    result = pthread\_join(thread1, NULL);  
    **if** (result != 0) {  
        fprintf(stderr, "can't join thread");  
        **return** EXIT\_FAILURE;  
    }  
    result = pthread\_join(thread2, NULL);  
    **if** (result != 0) {  
        fprintf(stderr, "can't join thread");  
        **return** EXIT\_FAILURE;  
    }  
    **return** 0;  
}



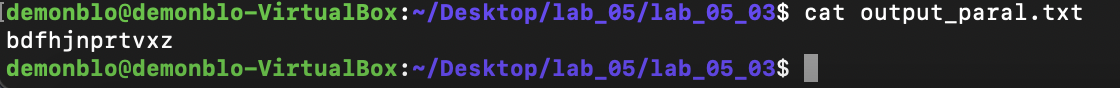


Рис. 12-13 результат работы третьей программы для нескольких потоков

## Связь структур:

## 

Рис. 14 - связь структур третьей программы

## Анализ работы третьей программы:

С помощью двух вызовов fopen() создаются два дескриптора файла в таблице файлов, открытых процессом, и две записи в системной таблице открытых файлов. Функция fprintf буферизует данные, следовательно запись производится в два разных буфера. Таким образом в первый буфер попадают нечетные символы, а во второй — четные. При выполнении fclose вызывается fflush, которая переписывает буфер в файл. Таким образом, при первом вызове fclose() осуществляется запись в файл данных из первого буфера, то есть нечетных символов. Затем при втором вызове fclose() осуществляется перезапись данных файла из второго буфера. В результате в файле содержатся данные из второго буфера.