**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 №5 по курсу

**«Операционные системы»**

**Тема** Буферизованный и небуферизованный ввод-вывод

**Студент** Климов И.С.

**Группа** ИУ7-62Б

**Оценка (баллы)**

**Преподаватели** Рязанова Н.Ю.

Москва — 2022 г.

## **Структура FILE**

В качестве операционной системы в данной лабораторной работе использовалась Ubuntu 20.04.

Описание структуры FILE представлено в файле /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

Описание структуры \_IO\_FILE представлено в файле /usr/include/x86\_64-linux-gnu/bits/types/FILE.h

/\* Copyright (C) 1991-2020 Free Software Foundation, Inc.

This file is part of the GNU C Library.

The GNU C Library is free software; you can redistribute it and/or

modify it under the terms of the GNU Lesser General Public

License as published by the Free Software Foundation; either

version 2.1 of the License, or (at your option) any later version.

The GNU C Library is distributed in the hope that it will be useful,

but WITHOUT ANY WARRANTY; without even the implied warranty of

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU

Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public

License along with the GNU C Library; if not, see

<https://www.gnu.org/licenses/>. \*/

#ifndef \_\_struct\_FILE\_defined

#define \_\_struct\_FILE\_defined 1

/\* Caution: The contents of this file are not part of the official

stdio.h API. However, much of it is part of the official \*binary\*

interface, and therefore cannot be changed. \*/

#if defined \_IO\_USE\_OLD\_IO\_FILE && !defined \_LIBC

error "\_IO\_USE\_OLD\_IO\_FILE should only be defined when building libc itself"

#endif

#if defined \_IO\_lock\_t\_defined && !defined \_LIBC

#error "\_IO\_lock\_t\_defined should only be defined when building libc itself"

#endif

#include <bits/types.h>

**struct** \_IO\_FILE;

**struct** \_IO\_marker;

**struct** \_IO\_codecvt;

**struct** \_IO\_wide\_data;

/\* During the build of glibc itself, \_IO\_lock\_t will already have been

defined by internal headers. \*/

#ifndef \_IO\_lock\_t\_defined

**typedef** **void** \_IO\_lock\_t;

#endif

/\* The tag name of this struct is \_IO\_FILE to preserve historic

C++ mangled names for functions taking FILE\* arguments.

That name should not be used in new code. \*/

**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

};

**struct** \_IO\_FILE\_complete

{

**struct** \_IO\_FILE \_file;

#endif

**\_\_off64\_t** \_offset;

/\* Wide character stream stuff. \*/

**struct** \_IO\_codecvt \*\_codecvt;

**struct** \_IO\_wide\_data \*\_wide\_data;

**struct** \_IO\_FILE \*\_freeres\_list;

**void** \*\_freeres\_buf;

**size\_t** \_\_pad5;

**int** \_mode;

/\* Make sure we don’t get into trouble again. \*/

**char** \_unused2[**15** \* **sizeof**(**int**) - **4** \* **sizeof**(**void** \*) - **sizeof**(**size\_t**)];

};

/\* These macros are used by bits/stdio.h and internal headers. \*/

#define \_\_getc\_unlocked\_body(\_fp) \

(\_\_glibc\_unlikely((\_fp)->\_IO\_read\_ptr >= (\_fp)->\_IO\_read\_end) \

? \_\_uflow(\_fp) \

: \*(unsigned char \*)(\_fp)->\_IO\_read\_ptr++)

#define \_\_putc\_unlocked\_body(\_ch, \_fp) \

(\_\_glibc\_unlikely((\_fp)->\_IO\_write\_ptr >= (\_fp)->\_IO\_write\_end) \

? \_\_overflow(\_fp, (unsigned char)(\_ch)) \

: (unsigned char)(\*(\_fp)->\_IO\_write\_ptr++ = (\_ch)))

#define \_IO\_EOF\_SEEN 0x0010

#define \_\_feof\_unlocked\_body(\_fp) (((\_fp)->\_flags & \_IO\_EOF\_SEEN) != 0)

#define \_IO\_ERR\_SEEN 0x0020

#define \_\_ferror\_unlocked\_body(\_fp) (((\_fp)->\_flags & \_IO\_ERR\_SEEN) != 0)

#define \_IO\_USER\_LOCK 0x8000

/\* Many more flag bits are defined internally. \*/

#endif

## **Первая программа**

**Код программы**, в которой открытие файла происходит в одном и том же потоке.

#include <fcntl.h>

#include <stdio.h>

#define BUF\_SIZE 20

**int** **main**(**void**)

{

**int** fd = open("./data/data.txt", 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 = **1**;

**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, "**\n**");

**return** **0**;

}

**Результат работы:**



**Код программы**, в которой открытие файла происходит в двух потоках.

#include <fcntl.h>

#include <pthread.h>

#include <stdio.h>

#include <unistd.h>

#define BUF\_SIZE 20

#define SLEEP\_TIME 2

**void** \***read\_buffer**(**void** \*args)

{

**int** fd = \*(**int**\*)args;

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

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

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

**int** flag = **0**;

**char** c;

**while** ((flag = fscanf(fs2, "%c", &c)) == **1**)

{

usleep(SLEEP\_TIME);

fprintf(stdout, "thread 2: %c**\n**", c);

}

**return** NULL;

}

**int** **main**(**void**)

{

setbuf(stdout, NULL);

**pthread\_t** **thread**;

**int** fd = open("./data/data.txt", O\_RDONLY);

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

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

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

pthread\_create(&**thread**, NULL, read\_buffer, (**void** \*)&fd);

**char** c;

**int** flag = **0**;

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

{

usleep(SLEEP\_TIME);

fprintf(stdout, "thread 1: %c**\n**", c);

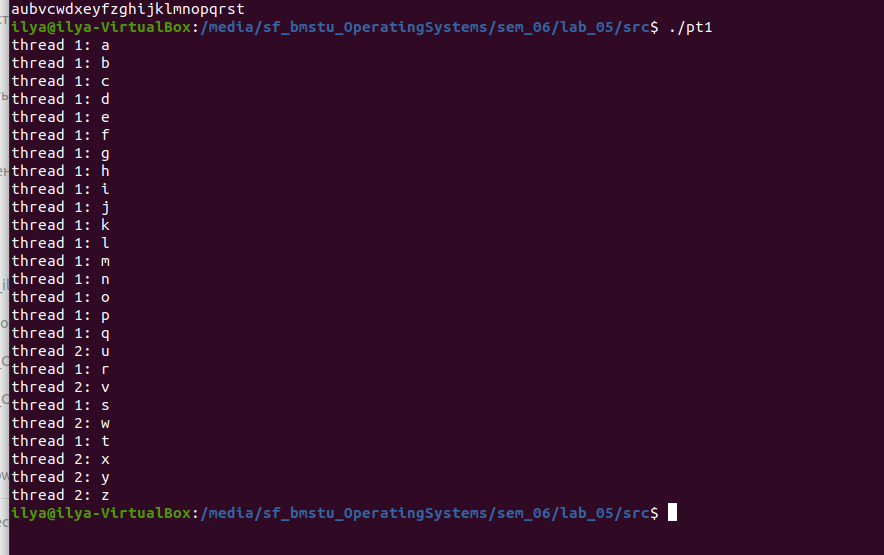
}

pthread\_join(**thread**, NULL);

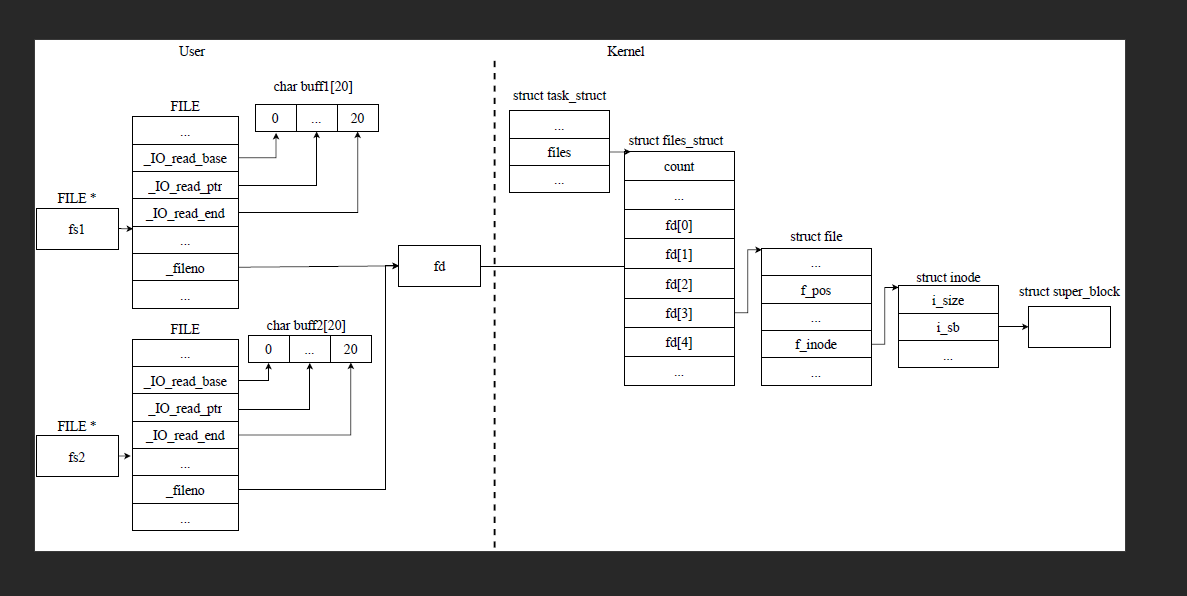
**return** **0**;

}

**Результат работы**:



**Связь структур**:



**Вывод**:

* функция open() создает файловый дескриптор открытого на чтение файла ./data/data.txt и запись в системной таблице открытых файлов, которая регистрирует смещение в файле и флаги состояния файла;
* функция fdopen() создает указатель на структуру FILE, fd содержит дескриптор, который вернула функция fopen();
* функция setvbuf() явно задает размер буфера в 20 байт и меняет тип буферизации (для fs1 и fs2) на полную;
* при первом вызове функции fscanf() в цикле (для fs1), buff1 будет заполнен полностью – первыми 20 символами (буквами алфавита). f\_pos в структуре struct\_file открытого файла увеличится на 20. При втором вызове fscanf() в цикле (для fs2) буфер buff2 будет заполнен оставшимися 6 символами (начиная с f\_pos);
* в цикле поочередно выводятся символы из buff1 и buff2;
* так же и для многопоточной реализации: в потоке, который первый получит квант, первый вызов fscanf() заполнит буфер и увеличит f\_pos в структуре struct\_file открытого файла, а оставшиеся 6 символов будут записаны в буфер, находящийся в другом потоке.

## **Вторая программа**

**Код программы**, в которой открытие файла происходит в одном и том же потоке.

#include <fcntl.h>

#include <unistd.h>

**int** **main**(**void**)

{

**int** fd1 = open("./data/data.txt", O\_RDONLY);

**int** fd2 = open("./data/data.txt", O\_RDONLY);

**int** rc1 = **1**, rc2 = **1**;

**char** c;

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

{

**if** ((rc1 = read(fd1, &c, **1**)) == **1**)

write(**1**, &c, **1**);

**if** ((rc2 = read(fd2, &c, **1**)) == **1**)

write(**1**, &c, **1**);

}

c = '\n';

write(**1**, &c, **1**);

**return** **0**;

}

**Результат работы**:



**Код программы**, в которой открытие файла происходит в двух потоках.

#include <fcntl.h>

#include <pthread.h>

#include <stdio.h>

#include <unistd.h>

**pthread\_mutex\_t** mutex;

**void** \***read\_buffer**(**void** \*args)

{

**int** fd2 = open("./data/data.txt", O\_RDONLY);

**char** c;

**int** err;

pthread\_mutex\_lock(&mutex);

**while** ((err = read(fd2, &c, **1**)) == **1**)

write(**1**, &c, **1**);

pthread\_mutex\_unlock(&mutex);

**return** NULL;

}

**int** **main**(**void**)

{

**int** fd1 = open("./data/data.txt", O\_RDONLY);

**pthread\_t** **thread**;

pthread\_create(&**thread**, NULL, read\_buffer, NULL);

**char** c;

**int** err;

pthread\_mutex\_lock(&mutex);

**while** ((err = read(fd1, &c, **1**)) == **1**)

write(**1**, &c, **1**);

pthread\_mutex\_unlock(&mutex);

pthread\_join(**thread**, NULL);

c = '\n';

write(**1**, &c, **1**);

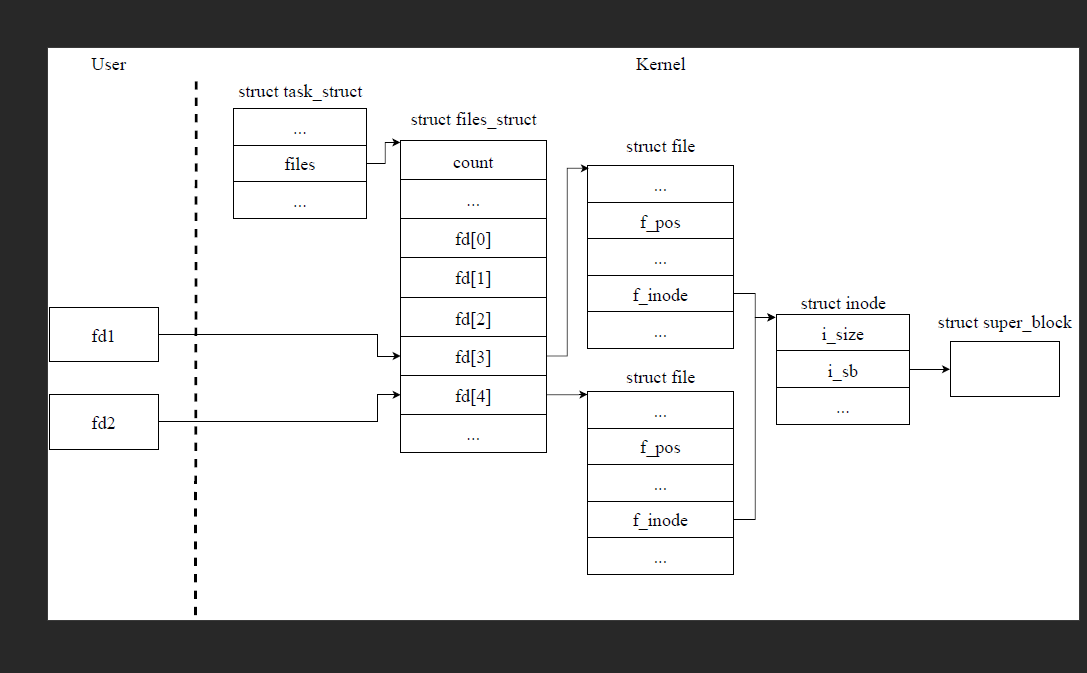
**return** **0**;

}

**Результат работы**:



**Связь структур**:



**Вывод**:

* функция open() создает файловый дескриптор, два раза для одного и того же файла, поэтому в программе существует две различные struct file, ссылающихся на один и тот же struct inode;
* из-за того, что структуры struct file разные, посимвольная печать просто дважды выведет содержимое файла в формате “aabbcc…” (в случае однопоточной реализации);
* в случае многопоточной реализации, вывод второго потока начнется позже (нужно время, для создание этого потока) и символы перемешаются.

## **Третья программа**

**Код программы**, в которой запись в файл происходит в одном и том же потоке.

#include <fcntl.h>

#include <stdio.h>

#include <unistd.h>

**int** **main**()

{

**FILE** \*f1 = fopen("./data/result\_3.txt", "w");

**FILE** \*f2 = fopen("./data/result\_3.txt", "w");

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

fprintf(c % **2** ? f1 : f2, "%c", c);

fclose(f2);

fclose(f1);

**return** **0**;

}

**Результат работы**:



**Код программы**, в которой запись в файл происходит в двух потоках.

#include <fcntl.h>

#include <pthread.h>

#include <stdio.h>

#include <unistd.h>

**void** \***write\_buffer**(**void** \*args)

{

**FILE** \*f = fopen("./data/result\_3\_pt.txt", "a");

**for** (**char** c = 'b'; c <= 'z'; c += **2**)

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

fclose(f);

**return** NULL;

}

**int** **main**()

{

**FILE** \*f1 = fopen("./data/result\_3\_pt.txt", "a");

**pthread\_t** **thread**;

**int** rc = pthread\_create(&**thread**, NULL, write\_buffer, NULL);

**for** (**char** c = 'a'; c <= 'z'; c += **2**)

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

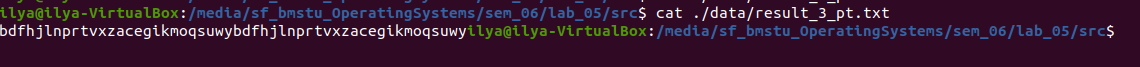
pthread\_join(**thread**, NULL);

fclose(f1);

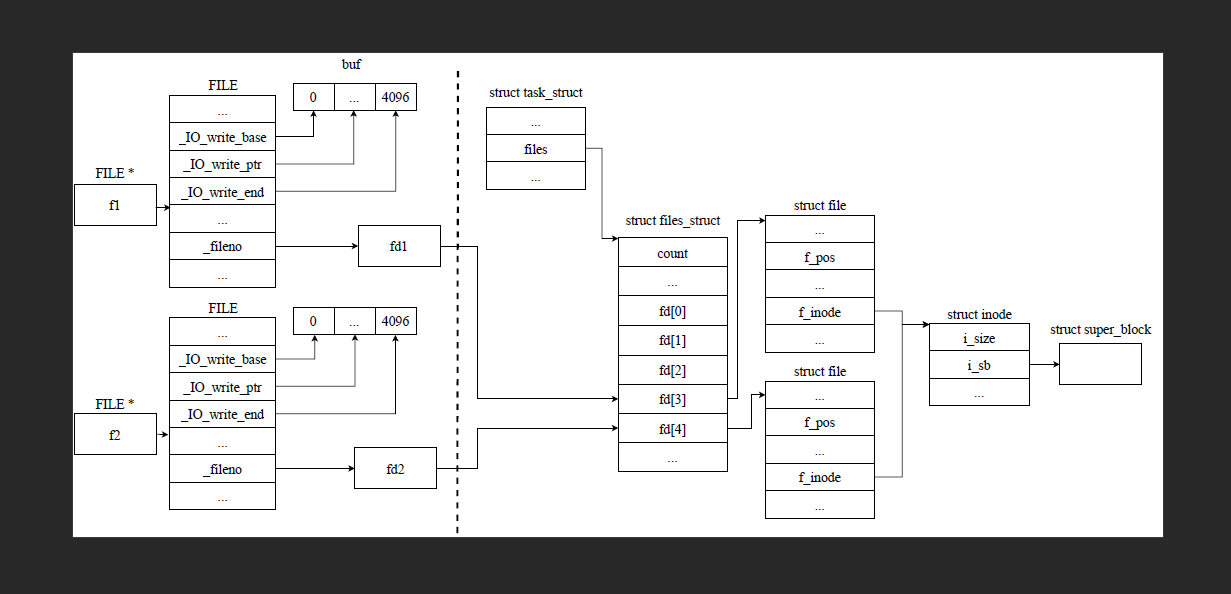
**return** **0**;

}

**Результат работы**:



**Связь структур**:



**Вывод**:

* функция open() вызывается два раза, в результате создается два файловых дескриптора в системной таблице открытых файлов и дескрипторов открытых файлов;
* функция fprintf() предоставляет буферизованный вывод;
* изначально информация пишется в буфер, а из буфера в файл если произошло одно из событий:
  + буфер полон;
  + вызвана функция fclose();
  + вызвана функция fflush();
* в данном случае информация в файл запишется в результате вызова функция fclose();
* каждая структура имеет свое поле f\_pos, эти изменяются независимо, так как было создано два файловых дескриптора. f\_pos обеих структур будет увеличиваться, начиная с 0, из-за чего содержимое файла будет выведено дважды;
* таким образом, информация записанная при первом вызове fclose() будет потеряна в результате второго вызова fclose() .
* в многопоточной реализации результат аналогичен – с помощью pthread\_join дожидаемся вызова fclose() для f2 в отдельном потоке и далее вызываем fclose() для f1.