

# **Отчёт по лабораторной работе №14**

**Именованные каналы**

Хусаинова Динара Айратовна

# Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Ход работы	8
4	Контрольные вопросы	12
5	Вывод	16

## Список иллюстраций

3.1	Создание файлов и каталога . . . . .	8
3.2	common.h . . . . .	9
3.3	server.c1 . . . . .	9
3.4	server.c2 . . . . .	10
3.5	client.c . . . . .	10
3.6	Makefile . . . . .	11
3.7	Компиляция . . . . .	11
3.8	Проверка работы файлов . . . . .	11

## List of Tables

# **1 Цель работы**

Приобретение практических навыков работы с именованными каналами.

## 2 Теоретическое введение

Одним из видов взаимодействия между процессами в операционных системах является обмен сообщениями. Под сообщением понимается последовательность байтов, передаваемая от одного процесса другому. В операционных системах типа UNIX есть 3 вида межпроцессорных взаимодействий:

общееюниксные (именованные каналы, сигналы), System V Interface Definition (SVID — разделяемая память, очередь сообщений, семафоры) и BSD (сокеты). Для передачи данных между неродственными процессами можно использовать механизм именованных каналов (named pipes). Данные передаются по принципу FIFO (First In First Out) (первым записан — первым прочитан), поэтому они называются также FIFO pipes или просто FIFO. Именованные каналы отличаются от неименованных наличием идентификатора канала, который представлен как специальный файл (соответственно имя именованного канала — это имя файла). Поскольку файл находится на локальной файловой системе, данное IPC используется внутри одной системы. Файлы именованных каналов создаются функцией `mkfifo(3)`. Первый параметр — имя файла, идентифицирующего канал, второй параметр — маска прав доступа к файлу. После создания файла канала процессы, участвующие в обмене данными, должны открыть этот файл либо для записи, либо для чтения. При закрытии файла сам канал продолжает существовать. Для того чтобы закрыть сам канал, нужно удалить его файл, например с помощью вызова `unlink(2)`. Рассмотрим работу именованного канала на примере системы клиент–сервер. Сервер создаёт канал, читает из него текст, посылаемый клиентом, и выводит его на терминал. Вызов функции `mkfifo()` создаёт файл канала (с

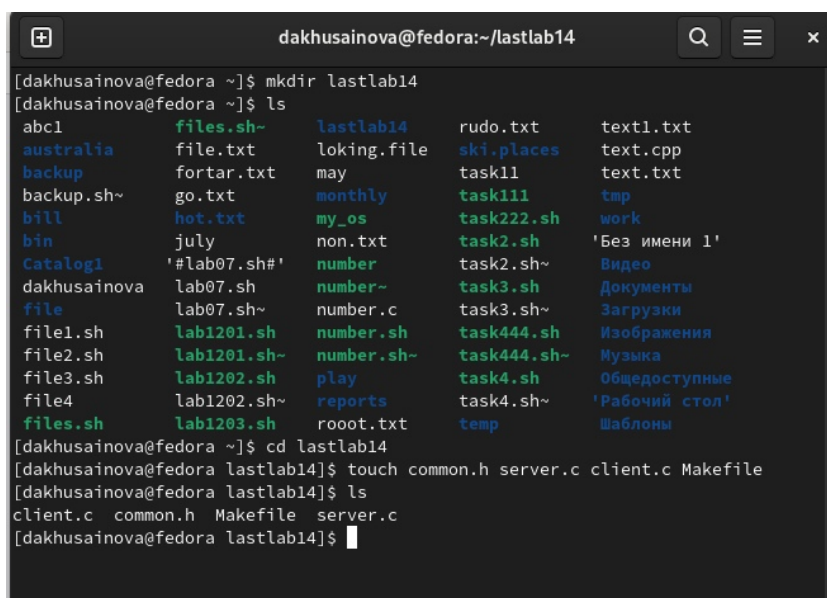
именем, заданным макросом `FIFO_NAME`).

В качестве маски доступа используется восьмеричное значение `0600`, разрешающее процессу с аналогичными реквизитами пользователя чтение и запись. Ждём сообщение от клиента. Сообщение читаем с помощью функции `read()` и печатаем на экран. После этого удаляется файл `FIFO_NAME` и сервер прекращает работу. Клиент открывает FIFO для записи как обычный файл:

Посылаем сообщение серверу с помощью функции `write()`. Для создания файла FIFO можно использовать более общую функцию `mknod(2)`, предназначенную для создания специальных файлов различных типов (FIFO, сокеты, файлы устройств и обычные файлы для хранения данных). 1 `mknod(FIFO_NAME, S_IFIFO | 0600, 0)`; Каналы представляют собой простое и удобное средство передачи данных, которое, однако, подходит не во всех ситуациях. Например, с помощью каналов довольно трудно организовать обмен асинхронными сообщениями между процессами.

## 3 Ход работы

1. Создаем необходимые файлы(рис. 3.1), после этого изучаем приведённые в тексте программы server.c и client.c. Взяв данные примеры за образец, напомним аналогичные программы, внеся изменения.

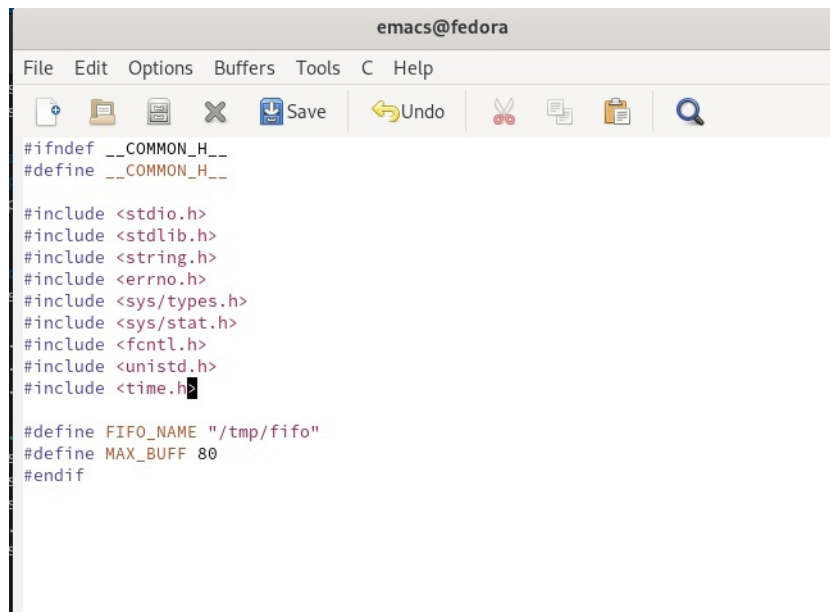


```
[dakhusainova@fedora ~]$ mkdir lastlab14
[dakhusainova@fedora ~]$ ls
abcl      files.sh~  lastlab14  rudo.txt   text1.txt
australia file.txt   looking.file ski.places  text.cpp
backup    fortar.txt may         task11      text.txt
backup.sh~ go.txt     monthly    task111     tmp
bill      hot.txt    my_os      task222.sh  work
bin       july      non.txt    task2.sh    'Без имени 1'
Catalog1  '#lab07.sh#' number     task2.sh~   Видео
dakhusainova lab07.sh  number~    task3.sh    Документы
file      lab07.sh~ number.c    task3.sh~   Загрузки
file1.sh  lab1201.sh number.sh   task444.sh  Изображения
file2.sh  lab1201.sh~ number.sh~  task444.sh~ Музыка
file3.sh  lab1202.sh play        task4.sh    Общедоступные
file4     lab1202.sh~ reports     task4.sh~   'Рабочий стол'
files.sh  lab1203.sh rooot.txt  temp        Шаблоны
[dakhusainova@fedora ~]$ cd lastlab14
[dakhusainova@fedora lastlab14]$ touch common.h server.c client.c Makefile
[dakhusainova@fedora lastlab14]$ ls
client.c common.h Makefile server.c
[dakhusainova@fedora lastlab14]$
```

Рис. 3.1: Создание файлов и каталога

В файле common.h добавляем стандартные заголовочные файлыunistd.h и time.h, необходимые для работы кодов других файлов(рис. 3.2).



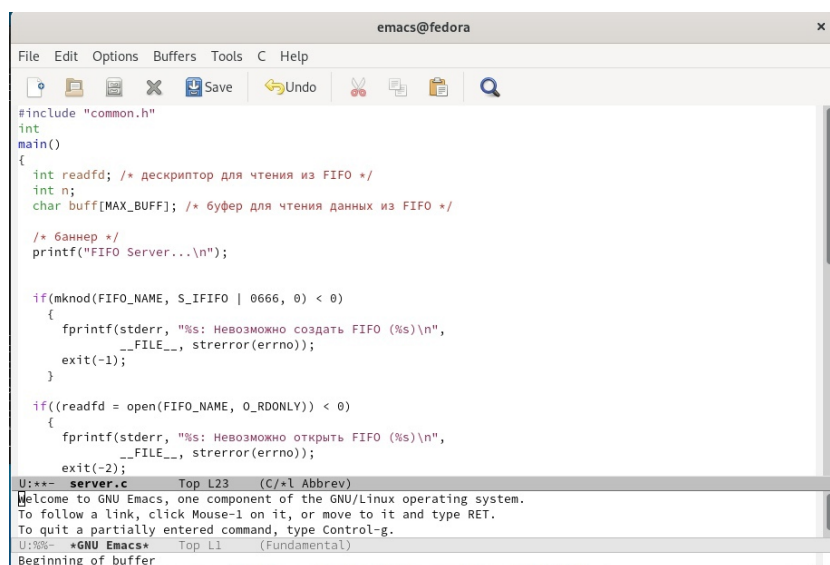


```
emacs@fedora
File Edit Options Buffers Tools C Help
Save Undo
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <time.h>

#define FIFO_NAME "/tmp/fifo"
#define MAX_BUFF 80
#endif
```

Рис. 3.2: common.h

В файл server.c добавляем цикл while для контроля за временем работы сервера(рис. 3.3,3.4).



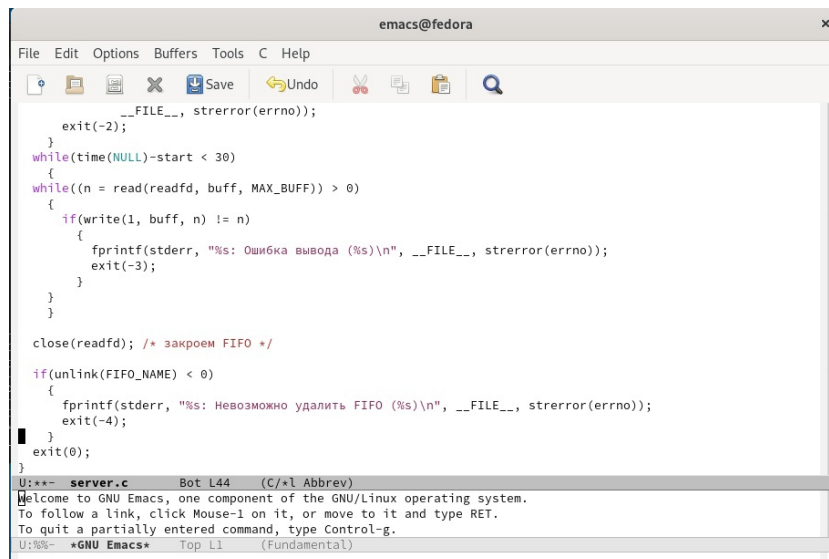
```
emacs@fedora
File Edit Options Buffers Tools C Help
Save Undo
#include "common.h"
int
main()
{
    int readfd; /* дескриптор для чтения из FIFO */
    int n;
    char buff[MAX_BUFF]; /* буфер для чтения данных из FIFO */

    /* баннер */
    printf("FIFO Server...\n");

    if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)
    {
        fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n",
            __FILE__, strerror(errno));
        exit(-1);
    }

    if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
    {
        fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
            __FILE__, strerror(errno));
        exit(-2);
    }
}
```

Рис. 3.3: server.c1



```
emacs@fedora
File Edit Options Buffers Tools C Help
[Icons] Save Undo [Icons]
exit(-2);
}
while(time(NULL)-start < 30)
{
while((n = read(readfd, buff, MAX_BUFF)) > 0)
{
if(write(1, buff, n) != n)
{
fprintf(stderr, "%s: Ошибка вывода (%s)\n", __FILE__, strerror(errno));
exit(-3);
}
}
}

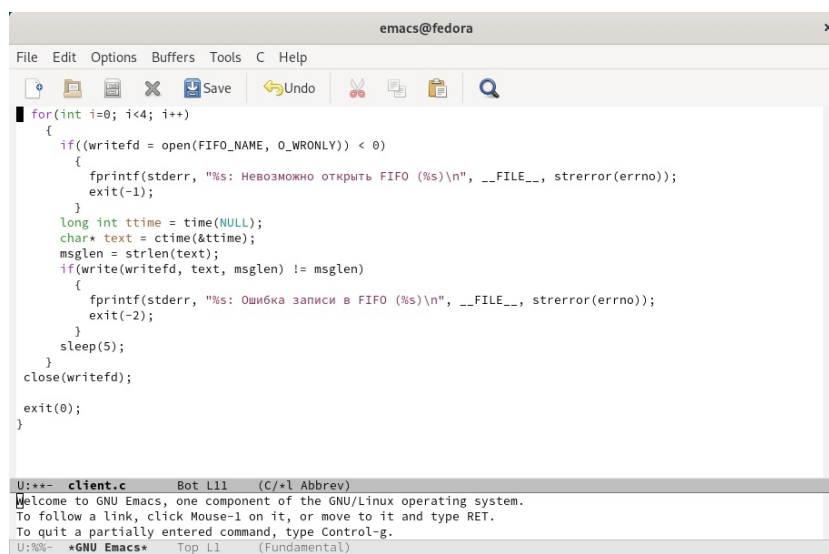
close(readfd); /* закроем FIFO */

if(unlink(FIFO_NAME) < 0)
{
fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n", __FILE__, strerror(errno));
exit(-4);
}
}
exit(0);
}

U:***- server.c Bot L44 (C/*l Abbrev)
Welcome to GNU Emacs, one component of the GNU/Linux operating system.
To follow a link, click Mouse-1 on it, or move to it and type RET.
To quit a partially entered command, type Control-g.
U:***- *GNU Emacs* Top L1 (Fundamental)
```

Рис. 3.4: server.c2

В файл client.c добавляем цикл, который отвечает за количество сообщений о текущем времени, которое получается в результате выполнения команд, и команду sleep(5) для приостановки работы клиента на 5 секунд(рис. 3.5).



```
emacs@fedora
File Edit Options Buffers Tools C Help
[Icons] Save Undo [Icons]
for(int i=0; i<4; i++)
{
if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
{
fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n", __FILE__, strerror(errno));
exit(-1);
}
long int ttime = time(NULL);
char* text = ctime(&ttime);
msglen = strlen(text);
if(write(writefd, text, msglen) != msglen)
{
fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n", __FILE__, strerror(errno));
exit(-2);
}
sleep(5);
}
close(writefd);
exit(0);
}

U:***- client.c Bot L11 (C/*l Abbrev)
Welcome to GNU Emacs, one component of the GNU/Linux operating system.
To follow a link, click Mouse-1 on it, or move to it and type RET.
To quit a partially entered command, type Control-g.
U:***- *GNU Emacs* Top L1 (Fundamental)
```

Рис. 3.5: client.c

2. Makefile оставляем без изменений( рис. 3.6).

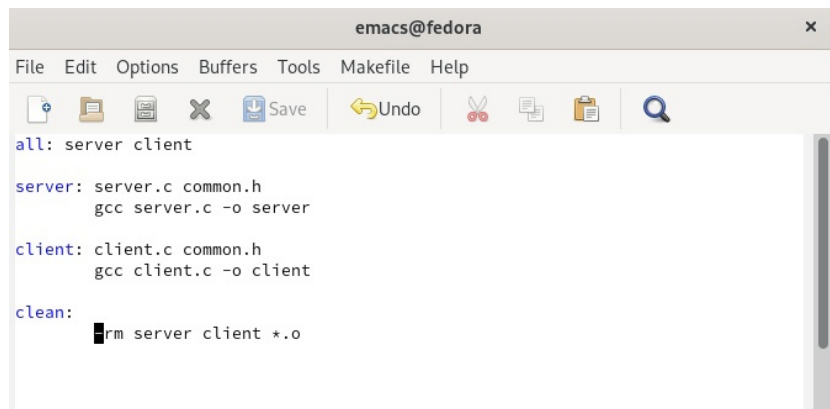


Рис. 3.6: Makefile

3. После написания кода, мы используем команду `make all`, чтобы скомпилировать все файлы( рис. 3.7).

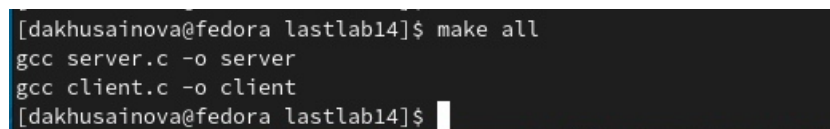


Рис. 3.7: Компиляция

5. Проверяем работу файлов. Открывает три консоли и запускаем: в первом терминале `server`, в остальных двух `client`. Также проверяем длительность работы сервера. Он завершил свою работу спустя 30 секунд. Если сервер завершит свою работу, не закрыв канал, то, когда мы будем запускать этот сервер снова, появится ошибка, так как у нас уже есть один канал( рис. 3.8).

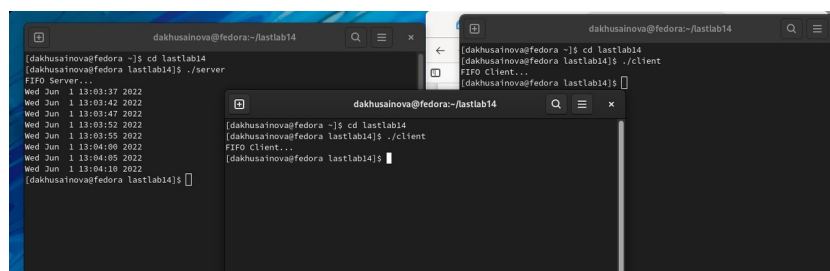


Рис. 3.8: Проверка работы файлов

## 4 Контрольные вопросы

1. В чем ключевое отличие именованных каналов от неименованных?

Именованные каналы отличаются от неименованных наличием идентификатора канала, который представлен как специальный файл (соответственно имя именованного канала – это имя файла). Поскольку файл находится на локальной файловой системе, данное IPC используется внутри одной системы.

2. Возможно ли создание неименованного канала из командной строки?

Чтобы создать неименованный канал из командной строки нужно использовать символ |, служащий для объединения двух и более процессов: процесс\_1 | процесс\_2 | процесс\_3...

3. Возможно ли создание именованного канала из командной строки?

Чтобы создать именованный канал из командной строки нужно использовать либо команду «mknod », либо команду «mkfifo».

4. Опишите функцию языка C, создающую неименованный канал.

Неименованный канал является средством взаимодействия между связанными процессами – родительским и дочерним. Родительский процесс создает канал при помощи системного вызова: «int pipe(int fd[2]);». Массив из двух целых чисел является выходным параметром этого системного вызова. Если вызов выполнен нормально, то этот массив содержит два файловых дескриптора. fd[0] является дескриптором для чтения из канала, fd[1] – дескриптором для записи в канал. Когда процесс порождает другой процесс, дескрипторы родительского процесса наследуются дочерним процессом, и, таким образом, прокладывается трубопровод между двумя процессами. Естественно, что один из процессов использует

канал только для чтения, а другой – только для записи. Поэтому, если, например, через канал должны передаваться данные из родительского процесса в дочерний, родительский процесс сразу после запуска дочернего процесса закрывает дескриптор канала для чтения, а дочерний процесс закрывает дескриптор для записи. Если нужен двунаправленный обмен данными между процессами, то родительский процесс создает два канала, один из которых используется для передачи данных в одну сторону, а другой – в другую.

5. Опишите функцию языка C, создающую именованный канал.

Файлы именованных каналов создаются функцией `mkfifo()` или функцией `mknod()`:

- «`int mkfifo(const char pathname, mode_t mode);`», где первый параметр – путь, где будет располагаться FIFO (имя файла, идентифицирующего канал), второй параметр определяет режим работы с FIFO (маска прав доступа к файлу),
- «`mknod(namefile, IFIFO | 0666, 0)`», где `namefile` – имя канала, `0666` – к каналу разрешен доступ на запись и на чтение любому запросившему процессу),
- «`int mknod(const char pathname, mode_t mode, dev_t dev);`». Функция `mkfifo()` создает канал и файл соответствующего типа. Если указанный файл канала уже существует, `mkfifo()` возвращает -1. После создания файла канала процессы, участвующие в обмене данными, должны открыть этот файл либо для записи, либо для чтения.

6. Что будет в случае прочтения из `fifo` меньшего числа байтов, чем находится в канале? Большого числа байтов?

При чтении меньшего числа байтов, чем находится в канале или FIFO, возвращается требуемое число байтов, остаток сохраняется для последующих чтений. При чтении большего числа байтов, чем находится в канале или FIFO, возвращается доступное число байтов. Процесс, читающий из канала, должен соответствующим образом обработать ситуацию, когда прочитано меньше, чем заказано.

7. Аналогично, что будет в случае записи в `fifo` меньшего числа байтов, чем позволяет буфер? Большого числа байтов?

Запись числа байтов, меньшего емкости канала или FIFO, гарантированно атомарно. Это означает, что в случае, когда несколько процессов одновременно

записывают в канал, порции данных от этих процессов не перемешиваются. При записи большего числа байтов, чем это позволяет канал или FIFO, вызов `write(2)` блокируется до освобождения требуемого места. При этом атомарность операции не гарантируется. Если процесс пытается записать данные в канал, не открытый ни одним процессом на чтение, процессу генерируется сигнал `SIGPIPE`, а вызов `write(2)` возвращает 0 с установкой ошибки (`errno=ERRPIPE`) (если процесс не установил обработки сигнала `SIGPIPE`, производится обработка по умолчанию – процесс завершается).

#### 8. . Могут ли два и более процессов читать или записывать в канал?

Количество процессов, которые могут параллельно присоединяться к любому концу канала, не ограничено. Однако если два или более процесса записывают в канал данные одновременно, каждый процесс за один раз может записать максимум `PIPE BUF` байтов данных. Предположим, процесс (назовем его А) пытается записать X байтов данных в канал, в котором имеется место для Y байтов данных. Если X больше, чем Y, только первые Y байтов данных записываются в канал, и процесс блокируется. Запускается другой процесс (например. В); в это время в канале появляется свободное пространство (благодаря третьему процессу, считывающему данные из канала). Процесс В записывает данные в канал. Затем, когда выполнение процесса А возобновляется, он записывает оставшиеся X-Y байтов данных в канал. В результате данные в канал записываются поочередно двумя процессами. Аналогичным образом, если два (или более) процесса одновременно попытаются прочитать данные из канала, может случиться так, что каждый из них прочитает только часть необходимых данных.

9. Опишите функцию `write` (тип возвращаемого значения, аргументы и логику работы). Что означает 1 (единица) в вызове этой функции в программе `server.c` (строка 42)?

Функция `write` записывает байты `count` из буфера `buffer` в файл, связанный с `handle`. Операции `write` начинаются с текущей позиции указателя на файл (указатель ассоциирован с заданным файлом). Если файл открыт для добавления,

операции выполняются в конец файла. После осуществления операций записи указатель на файл (если он есть) увеличивается на количество действительно записанных байтов. Функция `write` возвращает число действительно записанных байтов. Возвращаемое значение должно быть положительным, но меньше числа `count` (например, когда размер для записи `count` байтов выходит за пределы пространства на диске). Возвращаемое значение `-1` указывает на ошибку; `errno` устанавливается в одно из следующих значений: `EACCES` – файл открыт для чтения или закрыт для записи, `EBADF` – неверный `handle`-р файла, `ENOSPC` – на устройстве нет свободного места. Единица в вызове функции `write` в программе `server.c` означает идентификатор (дескриптор потока) стандартного потока вывода.

#### 10. Опишите функцию `strerror`.

Прототип функции `strerror`: `char * strerror( int errornum );`».

Функция `strerror` интерпретирует номер ошибки, передаваемый в функцию в качестве аргумента – `errornum`, в понятное для человека текстовое сообщение (строку). Откуда берутся эти ошибки? Ошибки эти возникают при вызове функций стандартных Си-библиотек. То есть хорошим тоном программирования будет – использование этой функции в паре с другой, и если возникнет ошибка, то пользователь или программист поймет, как исправить ошибку, прочитав сообщение функции `strerror`. Возвращенный указатель ссылается на статическую строку с ошибкой, которая не должна быть изменена программой. Дальнейшие вызовы функции `strerror` перезапишут содержание этой строки. Интерпретированные сообщения об ошибках могут различаться, это зависит от платформы и компилятора.

## **5 Вывод**

Я приобрела практические навыки работы с именованными каналами.