

Отчет по лабораторной работе №15

Тулеева Валерия, НБИбд-01-20

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук Кафедра прикладной
информатики и теории вероятностей

1 Именованные каналы

2 Оглавление:

1. Введение:
 - a) Цель работы
2. Описание результатов выполнения задания;
3. Библиография;
4. Вывод;
5. Контрольные вопросы.

3 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: общеюниксные (именованные каналы, сигналы), System V Interface Definition (SVID — разделяемая память, очередь сообщений, семафоры) и BSD (сокеты).

Источник

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).

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *pathname, mode_t mode);
```

Первый параметр — имя файла, идентифицирующего канал, второй параметр — маска прав доступа к файлу.

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. Источник

4 Цель работы:

Приобретение практических навыков работы с именованными каналами.

5 Описание результатов выполнения

Задания:

1. Создала файлы:

touch Makefile (рис 1.1):

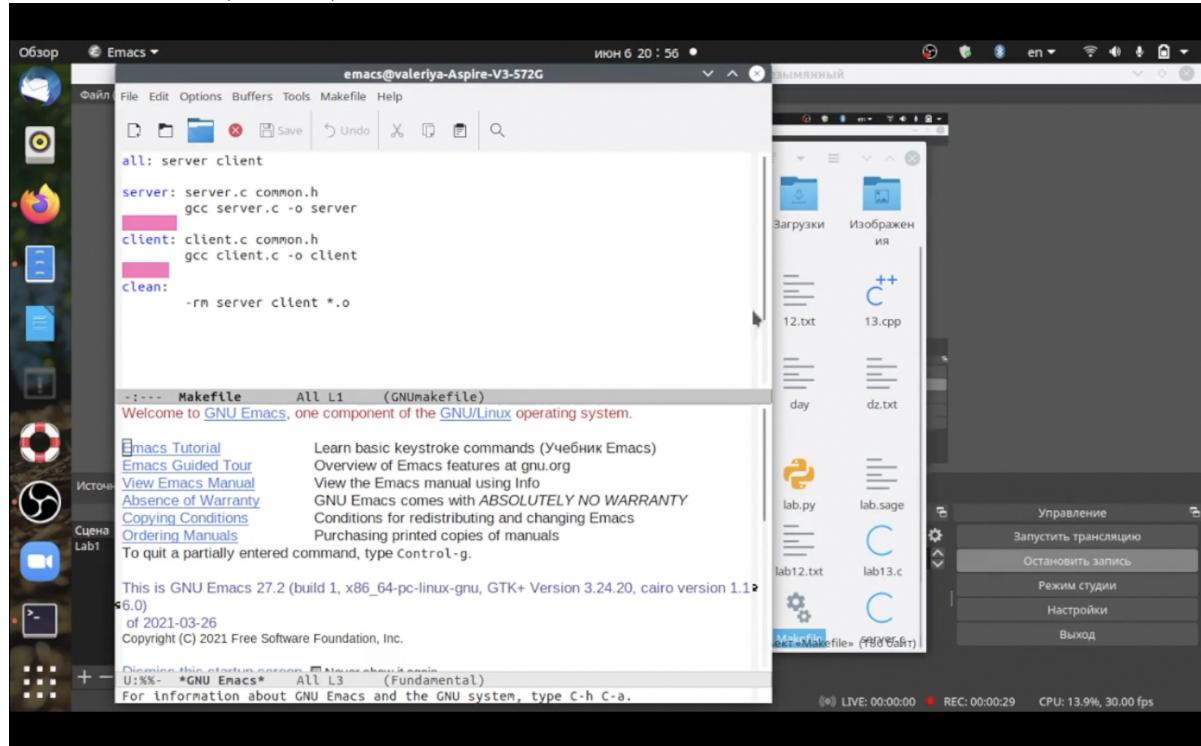


Рис 1.1. Makefile

touch server.c (рис 1.2):

```
12 main()
13 {
14     int readfd; /* дескриптор для чтения из FIFO */
15     int n;
16     char buff[MAX_BUFF]; /* буфер для чтения данных из FIFO */ /* баннер */
17
18     printf("FIFO Server...\n"); /* создаем файл FIFO с открытыми для всех правами доступа на чтение и запись*/
19
20     if(mknod(FIFO_NAME, S_IFIFO | 0666,0) < 0)
21     {
22         fprintf(stderr,"%s: Невозможно создать FIFO (%s)\n",
23                 __FILE__, strerror(errno));
24         exit(-1);/* откроем FIFO на чтение */
25
26     if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
27     {
28         fprintf(stderr,"%s: Невозможно открыть FIFO (%s)\n",
29                 __FILE__, strerror(errno));
30         exit(-2);
31
32     /* читаем данные из FIFO и выводим на экран */
33     while((n = read(readfd, buff, MAX_BUFF)) > 0)
34     {
35         if(write(1, buff, n) != n)
36         {
37             fprintf(stderr,"%s: Ошибка вывода (%s)\n",
38                     __FILE__, strerror(errno));
39             exit(-3);
40         }
41     }
42     close(readfd);/* закроем FIFO *//* удалим FIFO из системы */
43     if(unlink(FIFO_NAME) < 0)
44     {
45         fprintf(stderr,"%s: Невозможно удалить FIFO (%s)\n",
46                 __FILE__, strerror(errno));
47         exit(-4);
48     }
49     exit(0);
50 }
```

Рис 1.2. server.c

touch client.c (рис 1.3):

```
1/*
2 * client.c - реализация клиента
3 *
4 * чтобы запустить пример, необходимо:
5 * 1. запустить программу server на одной консоли;
6 * 2. запустить программу client на другой консоли.
7 */
8
9 #include"common.h"
10#define MESSAGE "Hello Server!!!\n"
11
12 int
13 main()
14 {
15     int writefd; /* дескриптор для записи в FIFO */
16     int msglen; /* баннер */
17
18     printf("FIFO Client...\n"); /* получим доступ к FIFO */
19
20     if((writefd=open(FIFO_NAME, O_WRONLY))<0)
21     {
22         fprintf(stderr,"%s: Невозможно открыть FIFO (%s)\n",
23                 __FILE__, strerror(errno));
24         exit(-1);/* передадим сообщение серверу */
25     msglen=strlen(MESSAGE);
26     if(write(writefd, MESSAGE, msglen)!=msglen)
27     {
28         fprintf(stderr,"%s: Ошибка записи в FIFO (%s)\n",
29                 __FILE__, strerror(errno));
30         exit(-2);/* закроем доступ к FIFO */
31     close(writefd);
32
33     exit(0);
34 }
```

Рис 1.3. client.c

touch common.h (рис 1.4):

```
1/*
2 * common.h - заголовочный файл со стандартными определениями
3 */
4
5#ifndef __COMMON_H__
6#define __COMMON_H__
7
8#include<stdio.h>
9#include<stdlib.h>
10#include<string.h>
11#include<errno.h>
12#include<sys/types.h>
13#include<sys/stat.h>
14#include<fcntl.h>
15
16#define FIFO_NAME          "/tmp/fifo"
17#define MAX_BUFF           80
18
19#endif/* __COMMON_H__ */
```

Рис 1.4. common.h

2. Далее я выполнила следующее задание:

Задание:

Изучите приведённые в тексте программы server.c и client.c. Взяв данные примеры за образец, напишите аналогичные программы, внеся следующие изменения:

1. Работает не 1 клиент, а несколько (например, два).
2. Клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд). Используйте функцию sleep() для приостановки работы клиента.
3. Сервер работает не бесконечно, а прекращает работу через некоторое время (например, 30 сек). Используйте функцию clock() для определения времени работы сервера. Что будет в случае, если сервер завершит работу, не закрыв канал?

Измененный Makefile (рис 2.1):

The screenshot shows the Emacs interface on a Linux desktop. The main window displays a Makefile with the following content:

```
all: server client

server: server.c common.h
        gcc server.c -o server

client: client.c common.h
        gcc client.c -o client

clean:
        -rm server client *.o
```

Below the code, there is a welcome message for GNU Emacs:

```
-- GNU Emacs 27.2 (build 1, x86_64-pc-linux-gnu, GTK+ Version 3.24.20, cairo version 1.16.0)
of 2021-06-04
Copyright (C) 2021 Free Software Foundation, Inc.
```

The status bar at the bottom indicates the file is a Makefile (152 bytes).

Рис 2.1. Makefile

Измененный server.c (рис 2.2):

The screenshot shows a Text Editor window with the file "server.c" open. The code is as follows:

```
1 #include "common.h"
2
3 int
4 main()
5 {
6     int readfd;
7     int n;
8     char buff[MAX_BUFF];
9     printf("FIFO Server...\n");
10    if(mknod(FIFO_NAME, S_IFIFO|0666,0) < 0)
11    {
12        fprintf(stderr,"%s: Невозможно создать FIFO (%s)\n",
13                __FILE__, strerror(errno));
14        exit(-1);
15    }
16
17    if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
18    {
19        fprintf(stderr,"%s: Невозможно открыть FIFO (%s)\n",
20                __FILE__, strerror(errno));
21        exit(-2);
22    }
23
24    clock_t now=time(NULL), start=time(NULL);
25    while((now-start)<30)
26    {
27        while((n = read(readfd, buff, MAX_BUFF)) > 0)
28        {
29            if(write(1, buff, n) != n)
30            {
31                fprintf(stderr,"%s: Ошибка вывода (%s)\n",
32                        __FILE__, strerror(errno));
33                exit(-3);
34            }
35        }
36    }
37    now=time(NULL);
38 }
```

The status bar at the bottom shows the file is being loaded from "/home/valeriya/server.c".

```
13     fprintf(stderr, "%s: невозможно создать FIFO (%s)\n",
14             __FILE__, strerror(errno));
15     exit(-1);
16 }
17
18 if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
19 {
20     fprintf(stderr, "%s: невозможно открыть FIFO (%s)\n",
21             __FILE__, strerror(errno));
22     exit(-2);
23 }
24
25 clock_t now=time(NULL), start=time(NULL);
26 while((now-start)<30)
27 {
28     while((n = read(readfd, buff, MAX_BUFFER)) > 0)
29     {
30         if(write(1, buff, n) != n)
31         {
32             fprintf(stderr, "%s: Ошибка вывода (%s)\n",
33                     __FILE__, strerror(errno));
34             exit(-3);
35         }
36     }
37     now=time(NULL);
38 }
39 printf("\n-----\nthe end\n-----\n");
40 close(readfd);
41
42 if(unlink(FIFO_NAME) < 0)
43 {
44     fprintf(stderr, "%s: невозможно удалить FIFO (%s)\n",
45             __FILE__, strerror(errno));
46     exit(-4);
47 }
48 exit(0);
49
50 }
```

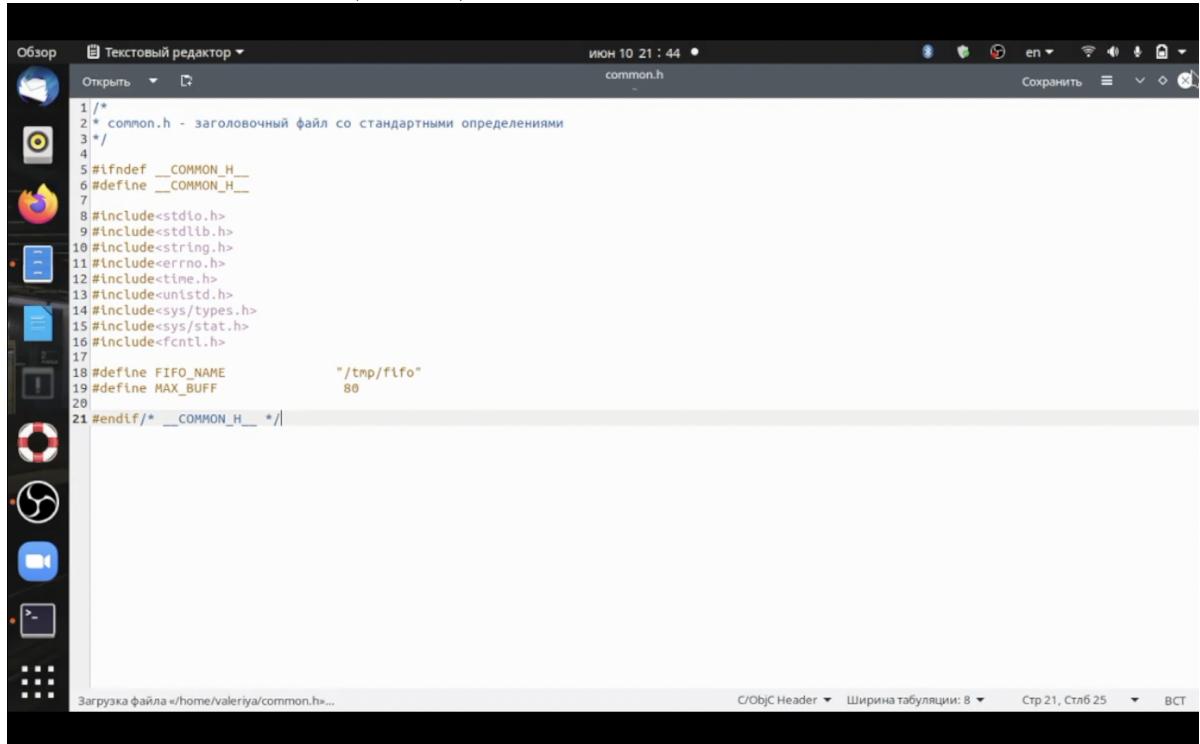
Рис 2.2. server.c

Измененный client.c (рис 2.3):

```
1 #include "common.h"
2 #define MESSAGE "Hello Server!!!\n"
3
4 int
5 main()
6 {
7     int writefd; /* дескриптор для записи в FIFO */
8     int msglen; /* банныер */
9     int number;
10    char message[10];
11    long long int T;
12    for(number=0; number<=5; ++number){
13        sleep(5);
14        T =(long long int) time(0);
15        sprintf (message, "%lli", T);
16        message[9]='\n';
17        printf("FIFO Client...\n");
18        if((writefd=open(FIFO_NAME, O_WRONLY))<0)
19        {
20            fprintf(stderr, "%s: невозможно открыть FIFO (%s)\n",
21                    __FILE__, strerror(errno));
22            exit(-1);
23        }
24        msglen=strlen(MESSAGE);
25        if(write(writefd, MESSAGE, msglen)!=msglen)
26        {
27            fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
28                    __FILE__, strerror(errno));
29            exit(-2);
30        }
31    }
32    close(writefd);
33 }
```

Рис 2.3. client.c

Измененный common.h (рис 2.4):



```
1/*
2 * common.h - заголовочный файл со стандартными определениями
3 */
4
5#ifndef __COMMON_H__
6#define __COMMON_H__
7
8#include<stdio.h>
9#include<stdlib.h>
10#include<string.h>
11#include<errno.h>
12#include<time.h>
13#include<unistd.h>
14#include<sys/types.h>
15#include<sys/stat.h>
16#include<fcntl.h>
17
18#define FIFO_NAME          "/tmp/fifo"
19#define MAX_BUFF           80
20
21#endif/* __COMMON_H__ */
```

Рис 2.4. coomon.h

3. Запуск: (рис 3.1)

Можно запустить или с помощью Makefile:

Ввести make

Или вот так:

gcc server.c -o server и gcc client.c -o client

./server и ./client

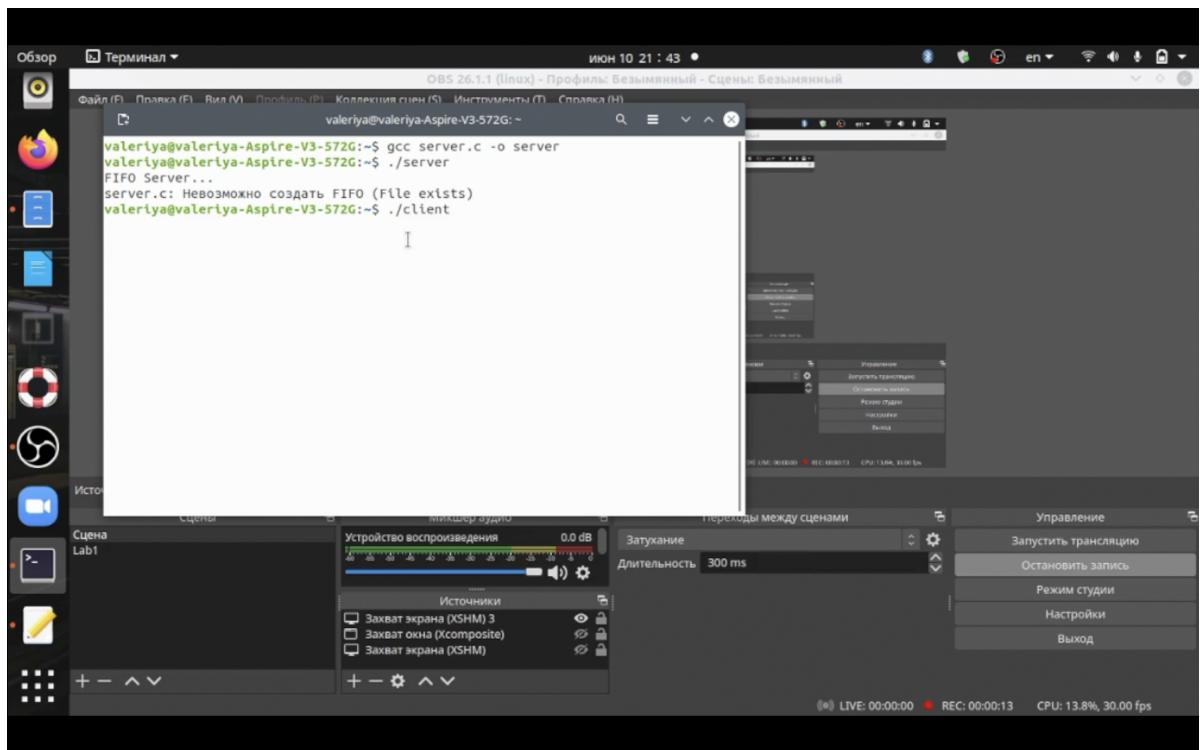


Рис 3.1. Запуск

6 Библиография:

1. Источник

7 Вывод:

В данной лабораторной работе, я приобрела практические навыки работы с именованными каналами.

8 Контрольные вопросы (лабораторная работа №15)

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).

2. Возможно ли создание неименованного канала из командной строки?

Неименованные каналы подобны именованным, но они в файловой системе не существуют. Они не имеют путевых имен, ассоциированных с ними, и все они и их следы исчезают после того, как последний файловый дескриптор, ссылающийся на них, закрывается. Они почти исключительно используются для межпроцессовых коммуникаций между дочерними и родительскими процессами либо между родственными процессами.

Оболочки применяют неименованные каналы для выполнения команд вроде `ls | head`. Процесс `ls` пишет в тот канал, из которого `head` читает свой ввод, выдавая ожидаемый пользователем результат.

Создание неименованного канала выполняется по двум файловым дескрипторам, один из которых доступен только для чтения, а второй — только для записи.

3. Возможно ли создание именованного канала из командной строки?

Файлы именованных каналов создаются функцией `mknod(3)`.

```
#include <sys/types.h>
#include <sys/stat.h>
```

```
int mkfifo(const char *pathname, mode_t mode);
```

Первый параметр — имя файла, идентифицирующего канал, второй параметр — маска прав доступа к файлу.

Вы можете создавать именованные каналы из командной строки и внутри программы.

4. Опишите функцию языка С, создающую неименованный канал.

Неименованный канал создается вызовом pipe, который заносит в массив int [2] два дескриптора открытых файлов. fd[0] – открыт на чтение, fd[1] – на запись (вспомните STDIN == 0, STDOUT == 1). Канал уничтожается, когда будут закрыты все файловые дескрипторы ссылающиеся на него.

В рамках одного процесса pipe смысла не имеет, передать информацию о нем в произвольный процесс нельзя (имени нет, а номера файловых дескрипторов в каждом процессе свои). Единственный способ использовать pipe – унаследовать дескрипторы при вызове fork (и последующем exec). После вызова fork канал окажется открытим на чтение и запись в родительском и дочернем процессе. Т.е. теперь на него будут ссылаться 4 дескриптора. Теперь надо определиться с направлением передачи данных – если надо передавать данные от родителя к потомку, то родитель закрывает дескриптор на чтение, а потомок – дескриптор на запись.

```
int fd[2];
char c;
pipe(fd);
if( fork() ) { //родитель
    ````close(fd[0]);````

    ````c=0;````

    ````while(write(fd[1],&c,1) >0)  {````

        ````c++;````
```

```
```}```

} else { //дочерний процесс

```dup2(fd[0],0); //подменили STDIN```

```close(fd[0]);```

```close(fd[1]);```

```execl("prog","prog",NULL); //запустили новую программу для которой STDIN = pipe```

}
```

Оставлять оба дескриптора незакрытыми плохо по двум причинам:

Родитель после записи не может узнать считал ли дочерний процесс данные, а если считал то сколько. Соответственно, если родитель попробует читать из pipe, то, вполне вероятно, он считает часть собственных данных, которые станут недоступными для потомка.

Если один из процессов завершился или закрыл свои дескрипторы, то второй этого не заметит, так как pipe на его стороне по-прежнему открыт на чтение и на запись.

Если надо организовать двунаправленную передачу данных, то можно создать два pipe.

5. Опишите функцию языка С, создающую именованный канал.

Именованный канал FIFO доступен как объект в файловой системе. При этом, до открытия объекта FIFO на чтение, собственно коммуникационного объекта не создаётся. После открытия открытия объекта FIFO в одном процессе на чтение, а в другом на запись, возникает ситуация полностью эквивалентная использованию неименованного канала.

Объект FIFO в файловой системе создаётся вызовом функции `int mkfifo(const char *pathname, mode_t mode);`

Основное отличие между pipe и FIFO - то, что pipe могут совместно использовать только процессы находящиеся в отношении родительский-дочерний, а FIFO может использовать любая пара процессов.

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

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

Мы можем писать в канал до тех пор пока есть место в буфере, если место в буфере кончится – процесс будет заблокирован на записи. Можем читать из канала пока есть данные в буфере, если данных нет – процесс будет заблокирован на чтении.

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

С точки зрения процессов, канал выглядит как пара открытых файловых дескрипторов – один на чтение и один на запись (можно больше, но неудобно).

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()` является частью UNIX-подобной системы ввода/вывода и не определена стандартом ANSI C. Функция `write()` переписывает `count` байт из буфера, на который указывает `buf` в файл, соответствующий дескриптору файла `handle`. Указателю положения в файле дается приращение на количество записанных байт.

Возвращаемым значением является количество действительно записанных байт. Если встретится ошибка, это количество может быть меньше, чем `count`. В случае ошибки возвращается `-1`.

Стандартный поток вывода имеет идентификатор 1.

10. Опишите функцию strerror.

Строковая функция strerror - функция языков C/C++, транслирующая код ошибки, который обычно хранится в глобальной переменной errno, в сообщение об ошибке, понятном человеку.