

Лабораторная работа №14

Именованные каналы

Медникова Екатерина Михайловна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Контрольные вопросы	13
5	Выводы	18

Список иллюстраций

3.1	Создание каталога и файлов	7
3.2	common.h	8
3.3	client.c	8
3.4	client.c	9
3.5	server.c	10
3.6	server.c	11
3.7	Makefile	11
3.8	Компиляция файлов	12
3.9	Компиляция файлов	12
3.10	Ошибка	12

Список таблиц

1 Цель работы

Приобретение практических навыков работы с именованными каналами.

2 Задание

Изучить приведённые в тексте программы `server.c` и `client.c`. Взяв данные примеры за образец, написать аналогичные программы, внося следующие изменения:

1. Работает не 1 клиент, а несколько (например, два).
2. Клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд). Использовать функцию `sleep()` для приостановки работы клиента.
3. Сервер работает не бесконечно, а прекращает работу через некоторое время (например, 30 сек). Использовать функцию `clock()` для определения времени работы сервера.

Что будет в случае, если сервер завершит работу, не закрыв канал?

3 Выполнение лабораторной работы

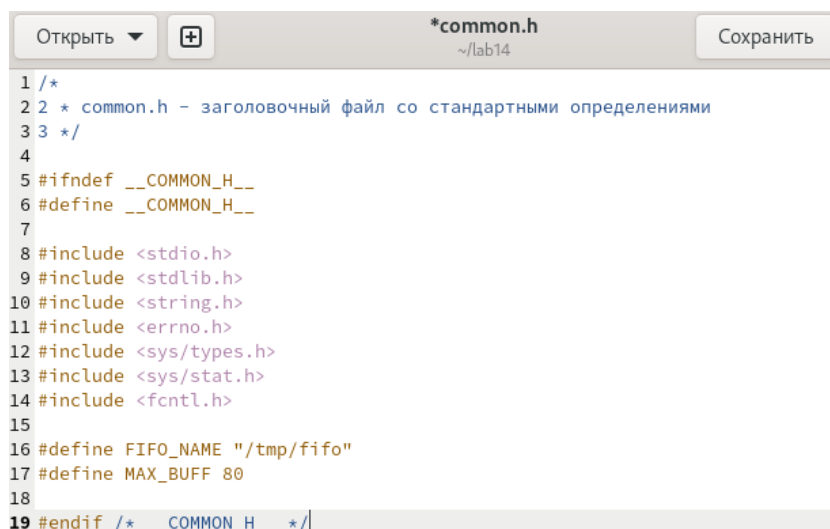
Изучила материал лабораторной работы. На основе примеров написала аналогичные программы, внося некоторые изменения.

1. Перед началом выполнения лабораторной работы создала каталог lab14 и файлы: common.h, server.c, client.c, Makefile.

```
[emmednikova@fedora ~]$ mkdir lab14
[emmednikova@fedora ~]$ cd lab14/
[emmednikova@fedora lab14]$ touch common.h server.c client.c Makefile
[emmednikova@fedora lab14]$ ls
client.c common.h Makefile server.c
[emmednikova@fedora lab14]$
```

Рис. 3.1: Создание каталога и файлов

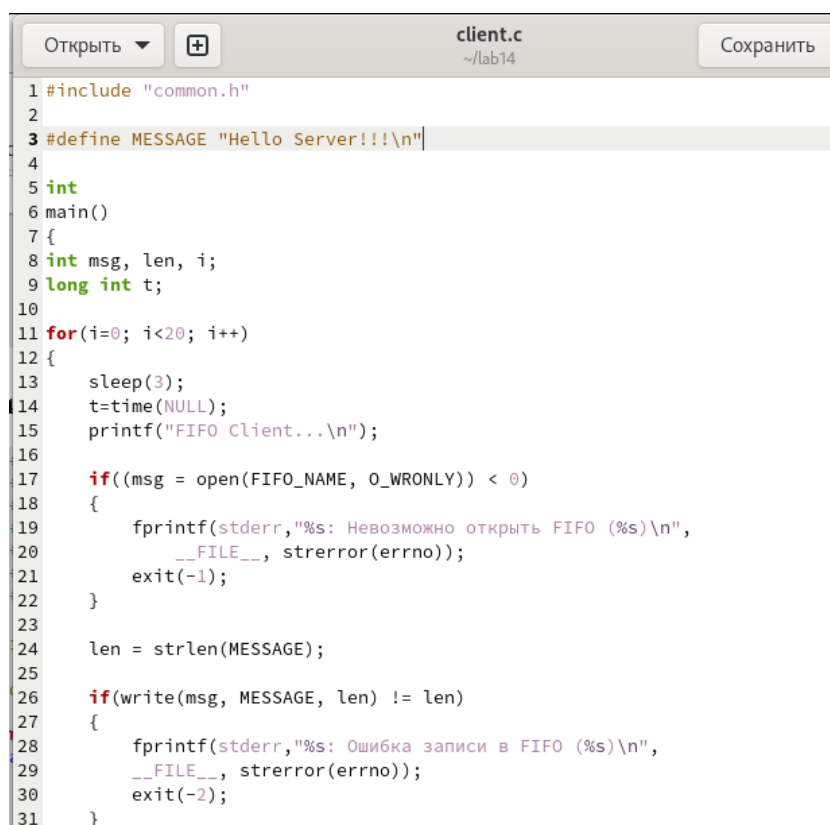
2. Вставила текст программы в файл common.h, ничего не меняя.



```
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__ */
```

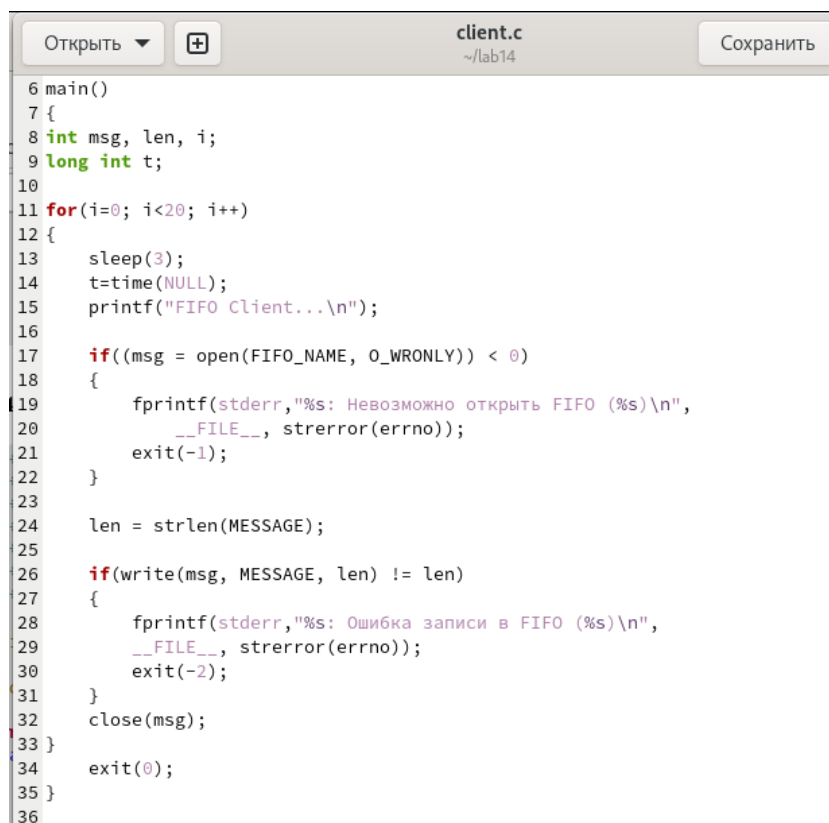
Рис. 3.2: common.h

3. Вставила текст программы в файл client.c, изменила его.



```
1 #include "common.h"
2
3 #define MESSAGE "Hello Server!!!\n"
4
5 int
6 main()
7 {
8     int msg, len, i;
9     long int t;
10
11     for(i=0; i<20; i++)
12     {
13         sleep(3);
14         t=time(NULL);
15         printf("FIFO Client...\n");
16
17         if((msg = open(FIFO_NAME, O_WRONLY)) < 0)
18         {
19             fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
20                     __FILE__, strerror(errno));
21             exit(-1);
22         }
23
24         len = strlen(MESSAGE);
25
26         if(write(msg, MESSAGE, len) != len)
27         {
28             fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
29                     __FILE__, strerror(errno));
30             exit(-2);
31         }
32     }
```

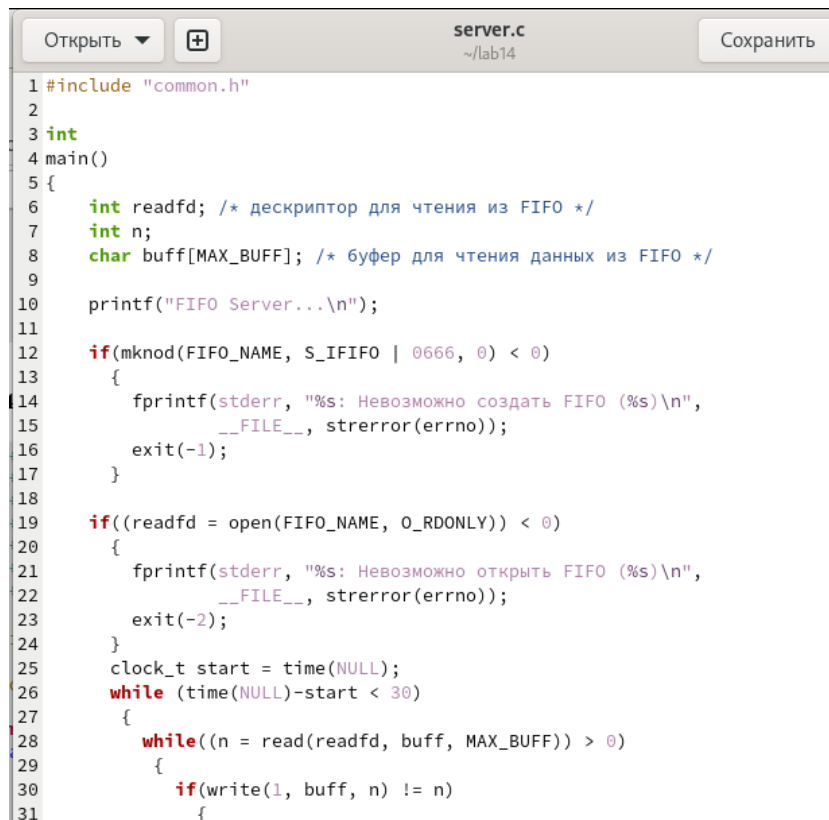
Рис. 3.3: client.c



```
6 main()
7 {
8     int msg, len, i;
9     long int t;
10
11     for(i=0; i<20; i++)
12     {
13         sleep(3);
14         t=time(NULL);
15         printf("FIFO client...\n");
16
17         if((msg = open(FIFO_NAME, O_WRONLY)) < 0)
18         {
19             fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
20                     __FILE__, strerror(errno));
21             exit(-1);
22         }
23
24         len = strlen(MESSAGE);
25
26         if(write(msg, MESSAGE, len) != len)
27         {
28             fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
29                     __FILE__, strerror(errno));
30             exit(-2);
31         }
32         close(msg);
33     }
34     exit(0);
35 }
36
```

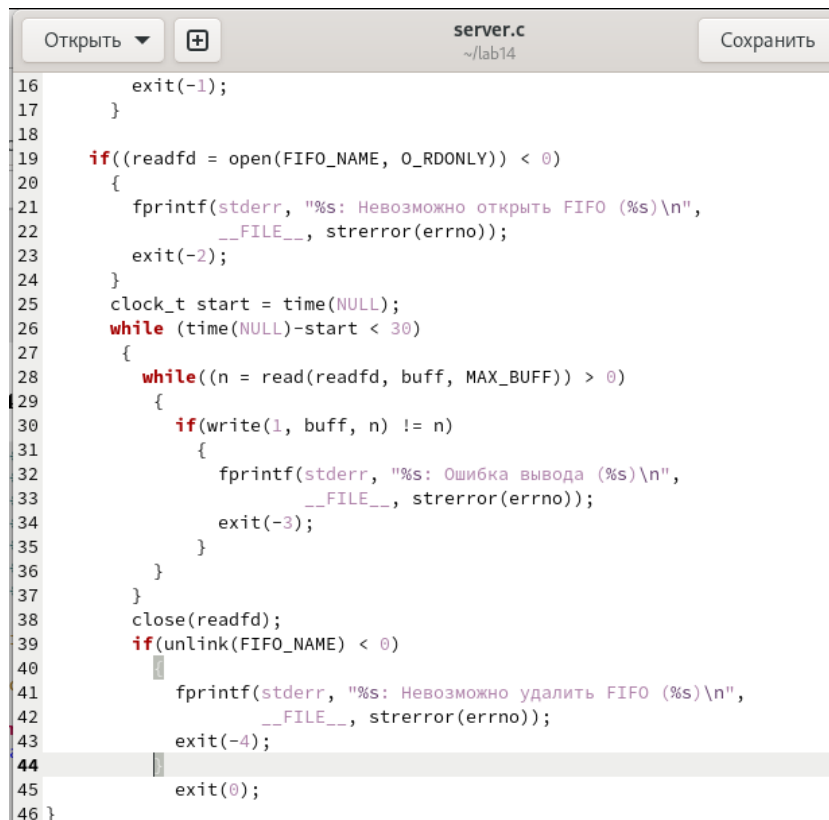
Рис. 3.4: client.c

4. Вставила текст программы в файл server.c, добавив некоторые коррективы.



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

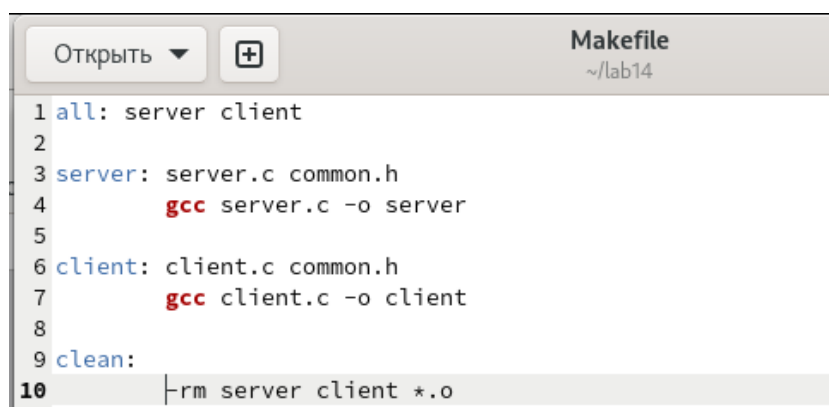
Рис. 3.5: server.c



```
16     exit(-1);
17 }
18
19 if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
20 {
21     fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
22             __FILE__, strerror(errno));
23     exit(-2);
24 }
25 clock_t start = time(NULL);
26 while (time(NULL)-start < 30)
27 {
28     while((n = read(readfd, buff, MAX_BUFF)) > 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 }
38 close(readfd);
39 if(unlink(FIFO_NAME) < 0)
40 {
41     fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n",
42             __FILE__, strerror(errno));
43     exit(-4);
44 }
45 exit(0);
46 }
```

Рис. 3.6: server.c

5. Вставила текст программы в файл Makefile, ничего не меняя.



```
1 all: server client
2
3 server: server.c common.h
4     gcc server.c -o server
5
6 client: client.c common.h
7     gcc client.c -o client
8
9 clean:
10     rm server client *.o
```

Рис. 3.7: Makefile

6. Далее делаем компиляцию файлов с помощью команды make all.

```

[emmednikova@fedora lab14]$ make all
gcc server.c -o server
server.c: В функции «main»:
server.c:25:23: предупреждение: неявная декларация функции «time» [-Wimplicit-fu
nction-declaration]
   25 |         clock_t start = time(NULL);
      |                             ^~~~
server.c:2:1: замечание: «time» is defined in header «<time.h>»; did you forget
to «#include <time.h>»?
   1 | #include "common.h"
   +++ |+#include <time.h>
   2 |
server.c:28:21: предупреждение: неявная декларация функции «read»; имелось в вид
е «fread»? [-Wimplicit-function-declaration]

```

Рис. 3.8: Компиляция файлов

```

gcc client.c -o client
client.c: В функции «main»:
client.c:13:5: предупреждение: неявная декларация функции «sleep» [-Wimplicit-fu
nction-declaration]
   13 |         sleep(3);
      |         ^~~~~
client.c:14:7: предупреждение: неявная декларация функции «time» [-Wimplicit-fu
nction-declaration]
   14 |         t=time(NULL);
      |         ^~~~
client.c:2:1: замечание: «time» is defined in header «<time.h>»; did you forget
to «#include <time.h>»?
   1 | #include "common.h"

```

Рис. 3.9: Компиляция файлов

7. При создании и проверке работы файлов каждый терминал (использовалось два терминала) вывел несколько сообщений, после 30 секунд работа сервера была завершена. Если сервер завершит работу, не закрывая канал, то при повторном запуске появится ошибка, так как уже существует один канал.

```

[emmednikova@fedora lab14]$ ./server
FIFO Server...
server.c: Невозможно создать FIFO (File exists)
[emmednikova@fedora lab14]$ ./server
FIFO Server...
server.c: Невозможно создать FIFO (File exists)
[emmednikova@fedora lab14]$

client.c:2:1: замечание: «time» is defined in header «<time.h>»; did you forget
to «#include <time.h>»?
   1 | #include "common.h"
   +++ |+#include <time.h>
   2 |
client.c:26:8: предупреждение: неявная декларация функции «fwrite» [-Wimplicit-fu
nction-declaration]
   26 |         if (fwrite(msg, MESSAGE, len) != len)
      |             ^~~~~~
client.c:32:5: предупреждение: неявная декларация функции «pclose» [-Wimplicit-fu
nction-declaration]
   32 |         pclose(msg);
      |         ^~~~~~
[emmednikova@fedora lab14]$ make all
make: Цепь «all» не требует выполнения команд.
[emmednikova@fedora lab14]$ ./client
FIFO Client...

```

Рис. 3.10: Ошибка

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.

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 создаёт канал при помощи системного вызова: `int pipe(int fd[2]);`. Массив из двух целых чисел является выходным параметром этого системного вызова. Если вызов выполнился 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 буд

«`mknod (namefile, IFIFO | 0666, 0)`», где `namefile` – имя канала, `0666` – к каналу р

«`int mknod(const char *pathname, mode_t mode, dev_t dev);`».

Функция `mkfifo()` создает канал и файл соответствующего типа. Если указанный файл

1. После создания файла канала процессы, участвующие в обмене данными, должны отк

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` перезапишет содержание этой строки. Интерпретированные сообщения об ошибках могут различаться, это зависит от платформы и компилятора.

5 Выводы

Приобрела практические навыки работы с именованными каналами.