

Отчёт по лабораторной работе 14

Именованные каналы

Ангелина Дмитриевна Чванова

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	12
5	Контрольные вопросы:	13
6	Библиография	17

Список иллюстраций

3.1	Создание файлов	7
3.2	Прграмма в файле common.h	7
3.3	Прграмма в файле client.c	8
3.4	Прграмма в файле client.c	8
3.5	Прграмма в файле server.c	9
3.6	Прграмма в файле server.c	9
3.7	Прграмма в Mikefile	10
3.8	Команда make all	10
3.9	Команда make all	11

Список таблиц

1 Цель работы

Приобретение практических навыков работы с именованными каналами.

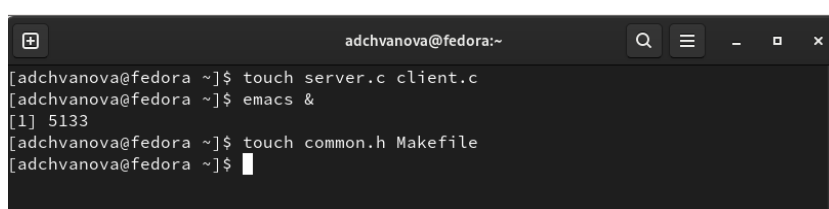
2 Задание

Изучите приведённые в тексте программы `server.c` и `client.c`. Взяв данные примеры за образец, напишите 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). 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?

3 Выполнение лабораторной работы

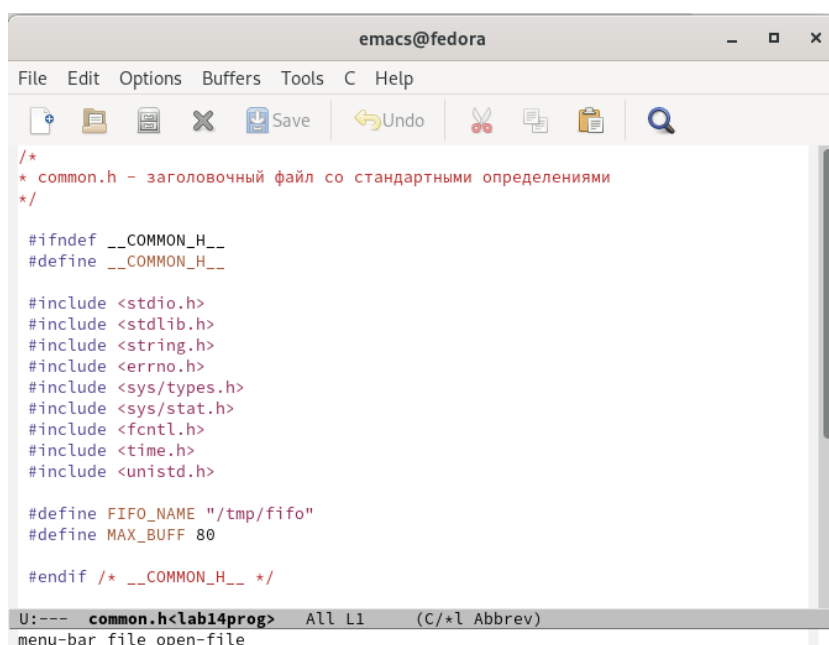
1). Я создала необходимые файлы с помощью команды «touch common.h server.c client.c Makefile» и открыла редактор emacs для их редактирования (рис. 3.1)



```
adchvanova@fedora:~  
[adchvanova@fedora ~]$ touch server.c client.c  
[adchvanova@fedora ~]$ emacs &  
[1] 5133  
[adchvanova@fedora ~]$ touch common.h Makefile  
[adchvanova@fedora ~]$
```

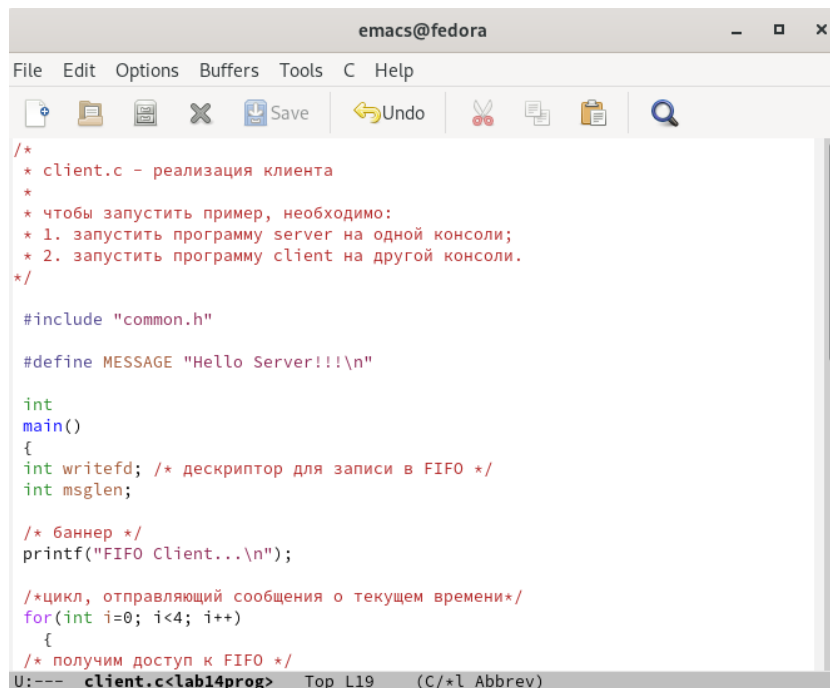
Рис. 3.1: Создание файлов

2). Далее я изменила коды программ, представленных в тексте лабораторной работы (рис. 3.2,3.3,3.4,3.5,3.6,3.7)



```
emacs@fedora  
File Edit Options Buffers Tools C Help  
+ Save Undo  
/*  
 * common.h - заголовочный файл со стандартными определениями  
 */  
  
#ifndef __COMMON_H__  
#define __COMMON_H__  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <errno.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <time.h>  
#include <unistd.h>  
  
#define FIFO_NAME "/tmp/fifo"  
#define MAX_BUFF 80  
  
#endif /* __COMMON_H__ */  
U:--- common.h<lab14prog> All L1 (C/*l Abbrev)  
menu-bar file open-file
```

Рис. 3.2: Программа в файле common.h



```
emacs@fedora
File Edit Options Buffers Tools C Help

/*
 * client.c - реализация клиента
 *
 * чтобы запустить пример, необходимо:
 * 1. запустить программу server на одной консоли;
 * 2. запустить программу client на другой консоли.
 */

#include "common.h"

#define MESSAGE "Hello Server!!!\n"

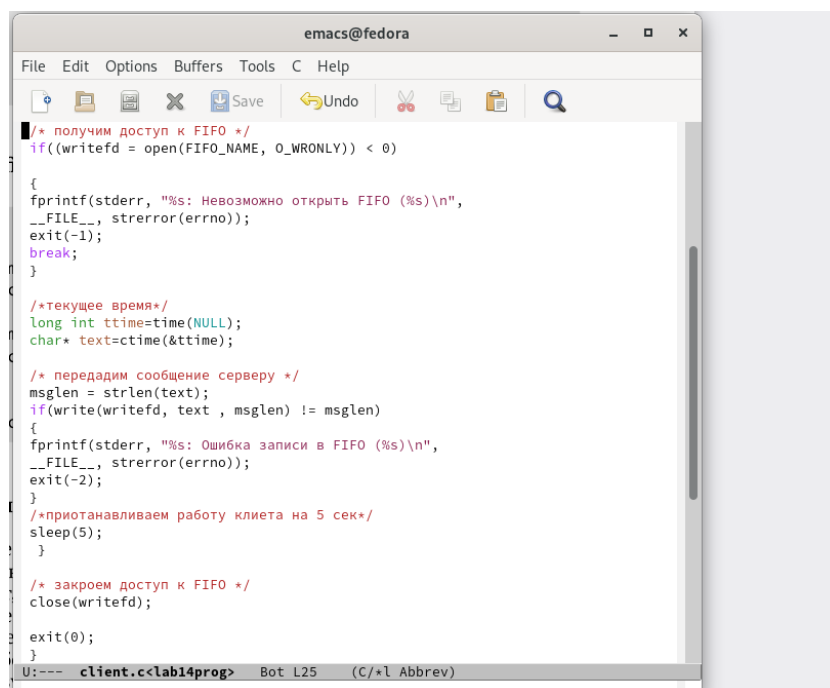
int
main()
{
    int writefd; /* дескриптор для записи в FIFO */
    int msglen;

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

    /*цикл, отправляющий сообщения о текущем времени*/
    for(int i=0; i<4; i++)
    {
        /* получим доступ к FIFO */
    }
}

U:--- client.c<lab14prog> Top L19 (C/*l Abbrev)
```

Рис. 3.3: Программа в файле client.c



```
emacs@fedora
File Edit Options Buffers Tools C Help

/* получим доступ к FIFO */
if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
{
    fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
        __FILE__, strerror(errno));
    exit(-1);
    break;
}

/*текущее время*/
long int ttime=time(NULL);
char* text=ctime(&ttime);

/* передадим сообщение серверу */
msglen = strlen(text);
if(write(writefd, text , msglen) != msglen)
{
    fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
        __FILE__, strerror(errno));
    exit(-2);
}

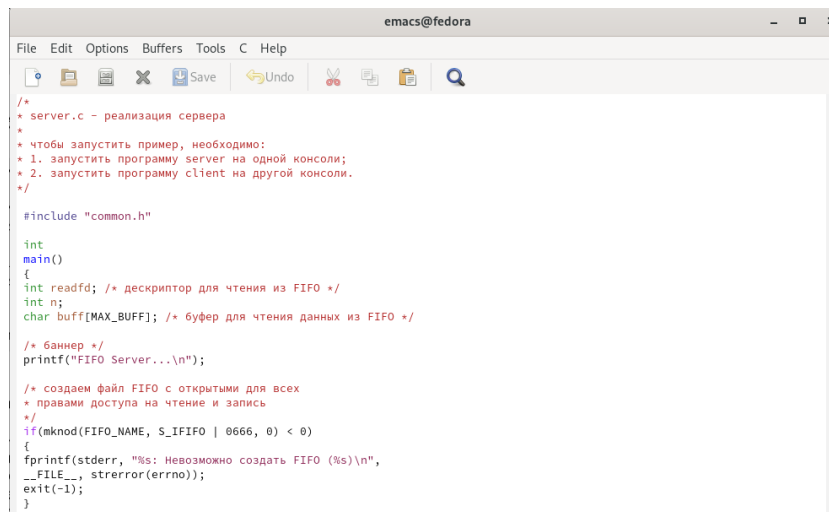
/*приотнавливаем работу клиента на 5 сек*/
sleep(5);
}

/* закроем доступ к FIFO */
close(writefd);

exit(0);
}

U:--- client.c<lab14prog> Bot L25 (C/*l Abbrev)
```

Рис. 3.4: Программа в файле client.c



```
emacs@fedora
File Edit Options Buffers Tools C Help
[Icons] Save Undo [Icons]

/*
 * server.c - реализация сервера
 *
 * чтобы запустить пример, необходимо:
 * 1. запустить программу server на одной консоли;
 * 2. запустить программу client на другой консоли.
 */

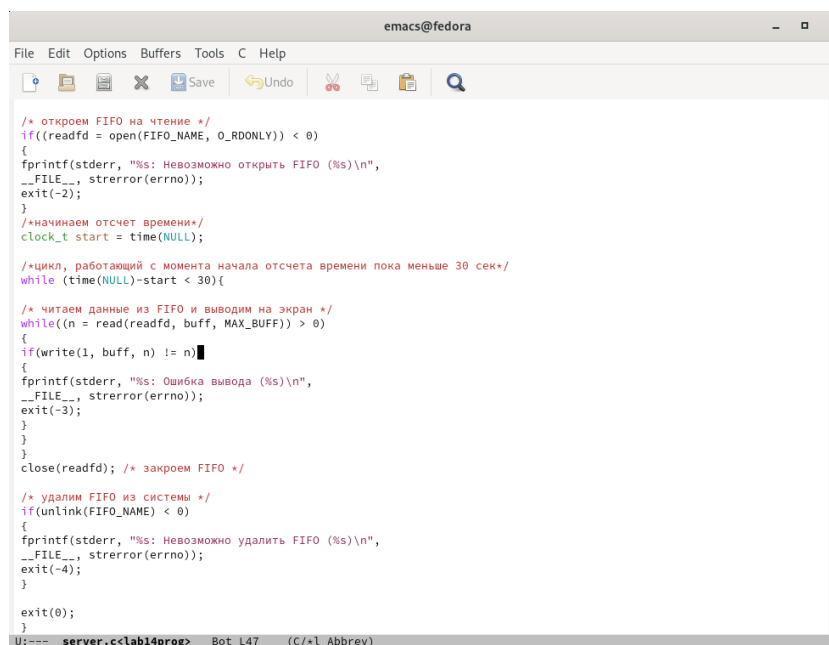
#include "common.h"

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");

    /* создаем файл FIFO с открытыми для всех
     * правами доступа на чтение и запись
     */
    if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)
    {
        fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n",
            _FILE_, strerror(errno));
        exit(-1);
    }
}
```

Рис. 3.5: Программа в файле server.c



```
emacs@fedora
File Edit Options Buffers Tools C Help
[Icons] Save Undo [Icons]

/* откроем FIFO на чтение */
if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
{
    fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
        _FILE_, strerror(errno));
    exit(-2);
}

/* начинаем отсчет времени */
clock_t start = time(NULL);

/* цикл, работающий с момента начала отсчета времени пока меньше 30 сек */
while (time(NULL) - start < 30){

    /* читаем данные из FIFO и выводим на экран */
    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 */

    /* удалим FIFO из системы */
    if(unlink(FIFO_NAME) < 0)
    {
        fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n",
            _FILE_, strerror(errno));
        exit(-4);
    }
}

exit(0);
}

U:--- server.c<lab14prog> Bot L47 (C/*l Abbrev)
```

Рис. 3.6: Программа в файле server.c

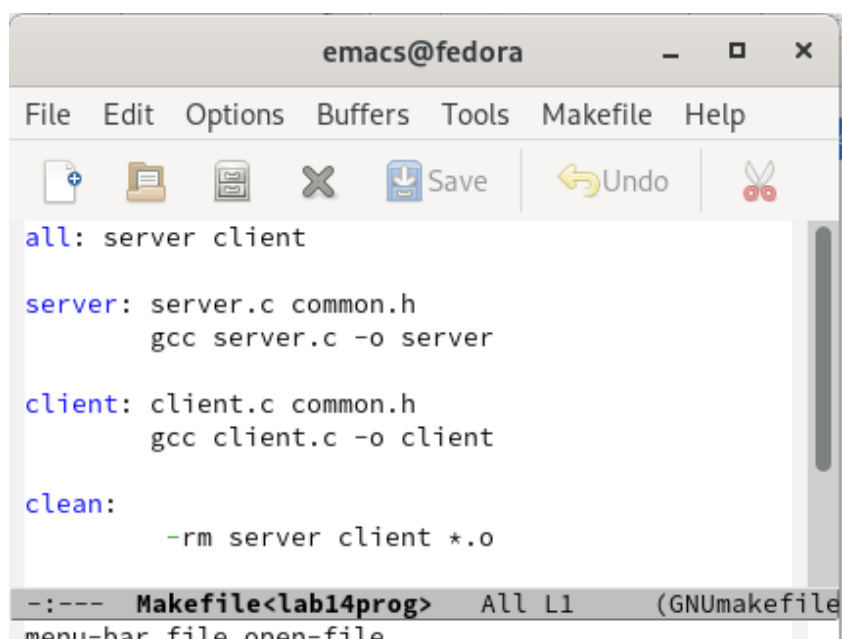


Рис. 3.7: Программа в Makefile

3).Используя команду «make all», скомпилировала необходимые файлы (рис. 3.8).

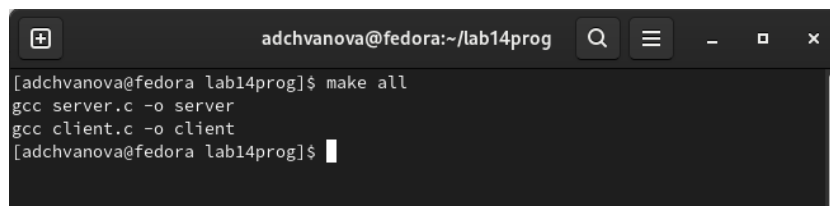
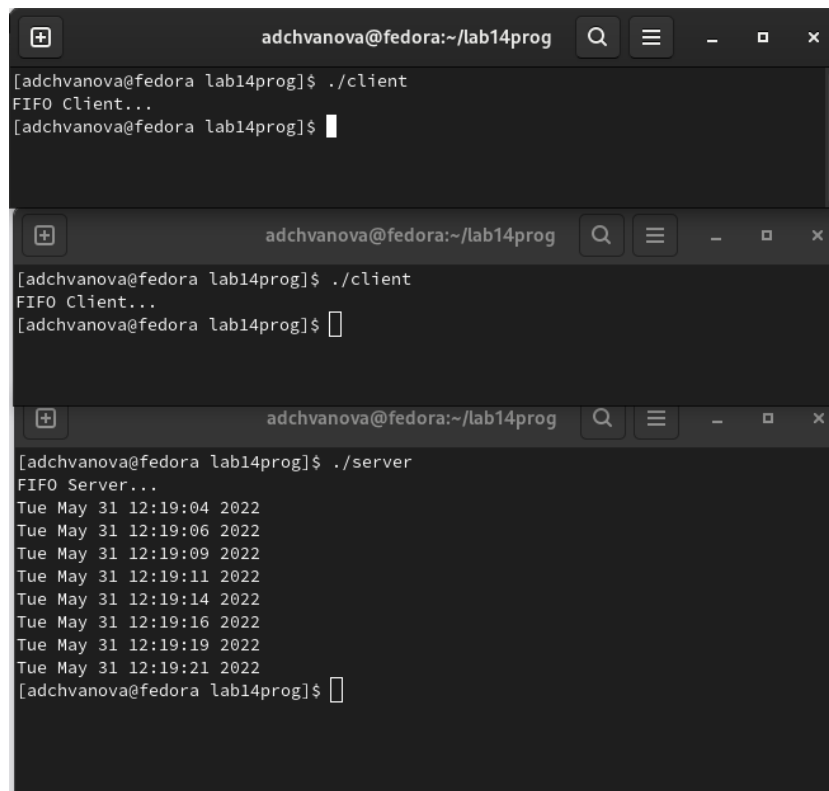


Рис. 3.8: Команда make all

Открыла 3 консоли (терминала) и запустила: «./server», 2 раза – «./client». Каждый терминал-клиент вывел по 4 сообщения. (рис. 3.9)



The image shows three terminal windows stacked vertically, all with the title bar 'adchvanova@fedora:~/lab14prog'.
The top window shows the command `./client` being executed, resulting in the output 'FIFO Client...'.
The middle window also shows the command `./client` being executed, resulting in the output 'FIFO Client...'.
The bottom window shows the command `./server` being executed, resulting in the output 'FIFO Server...' followed by seven lines of timestamps: 'Tue May 31 12:19:04 2022' through 'Tue May 31 12:19:21 2022'.

```
adchvanova@fedora:~/lab14prog
[adchvanova@fedora lab14prog]$ ./client
FIFO Client...
[adchvanova@fedora lab14prog]$

adchvanova@fedora:~/lab14prog
[adchvanova@fedora lab14prog]$ ./client
FIFO Client...
[adchvanova@fedora lab14prog]$

adchvanova@fedora:~/lab14prog
[adchvanova@fedora lab14prog]$ ./server
FIFO Server...
Tue May 31 12:19:04 2022
Tue May 31 12:19:06 2022
Tue May 31 12:19:09 2022
Tue May 31 12:19:11 2022
Tue May 31 12:19:14 2022
Tue May 31 12:19:16 2022
Tue May 31 12:19:19 2022
Tue May 31 12:19:21 2022
[adchvanova@fedora lab14prog]$
```

Рис. 3.9: Команда make all

Если сервер завершит свою работу, не закрыв канал, то, когда мы будем запускать этот сервер снова, появится ошибка «Невозможно создать FIFO», так как у нас уже есть один канал.

4 Выводы

Мы приобрели практические навыки работы с именованными каналами.

5 Контрольные вопросы:

1). 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). Чтобы создать неименованный канал из командной строки нужно использовать символ |, служащий для объединения двух и более процессов: процесс_1 | процесс_2 | процесс_3...

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

4). 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). Файлы именованных каналов создаются функцией `mkfifo()` или функцией `mknod`: 1. «`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), 2. «`mknod (namefile, IFIFO | 0666, 0)`», где `namefile` – имя канала, 0666 – к каналу разрешен доступ на запись и на чтение любому запросившему процессу), 3. «`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). 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). 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). 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). 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). Прототип функции strerror: «char * strerror(int errornum);». Функция strerror

интерпретирует номер ошибки, передаваемый в функцию в качестве аргумента `-errno`, 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.

6 Библиография

1. Программное обеспечение GNU/Linux. Лекция 10. Минимальный набор знаний (Г. Курячий, МГУ)
2. Программное обеспечение GNU/Linux. Лекция 12. Выбор дистрибутива (Г. Курячий, МГУ)
3. Электронный ресурс: <https://habr.com/ru/post/122108/>
4. Электронный ресурс: https://www.opennet.ru/docs/RUS/linux_parallel/node17.html