

# РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

---

## ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 15

дисциплина: Операционные системы

**Студент:**

Афтаева Ксения Васильевна

**Преподаватель:**

Велиева Т.В.

**Группа:** НПИбд-01-20

---

МОСКВА 2021 г.

### Цель работы:

Приобретение практических навыков работы с **именованными каналами**

### Задачи:

1. Изучить теоретический материал
2. Внести изменения в программы, представленные в описании лабораторной
3. Ответить на контрольные вопросы.

### Объект и предмет исследования:

Программирование в оболочке **ОС UNIX/Linux**

### Техническое оснащение:

Ноутбук, на котором установлена виртуальная машина с Linux

### Теоретические вводные данные: [1]

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*:

- общедюниксные (именованные каналы, сигналы)
- System V Interface Definition (SVID — разделяемая память, очередь сообщений, семафоры)
- 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)`

```
#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.

Вызов функции `mkfifo()` создаёт файл канала (с именем, заданным макросом `FIFO_NAME`):

```
mkfifo(FIFO_NAME, 0600)
```

## Условные обозначения и символы:

Не используются

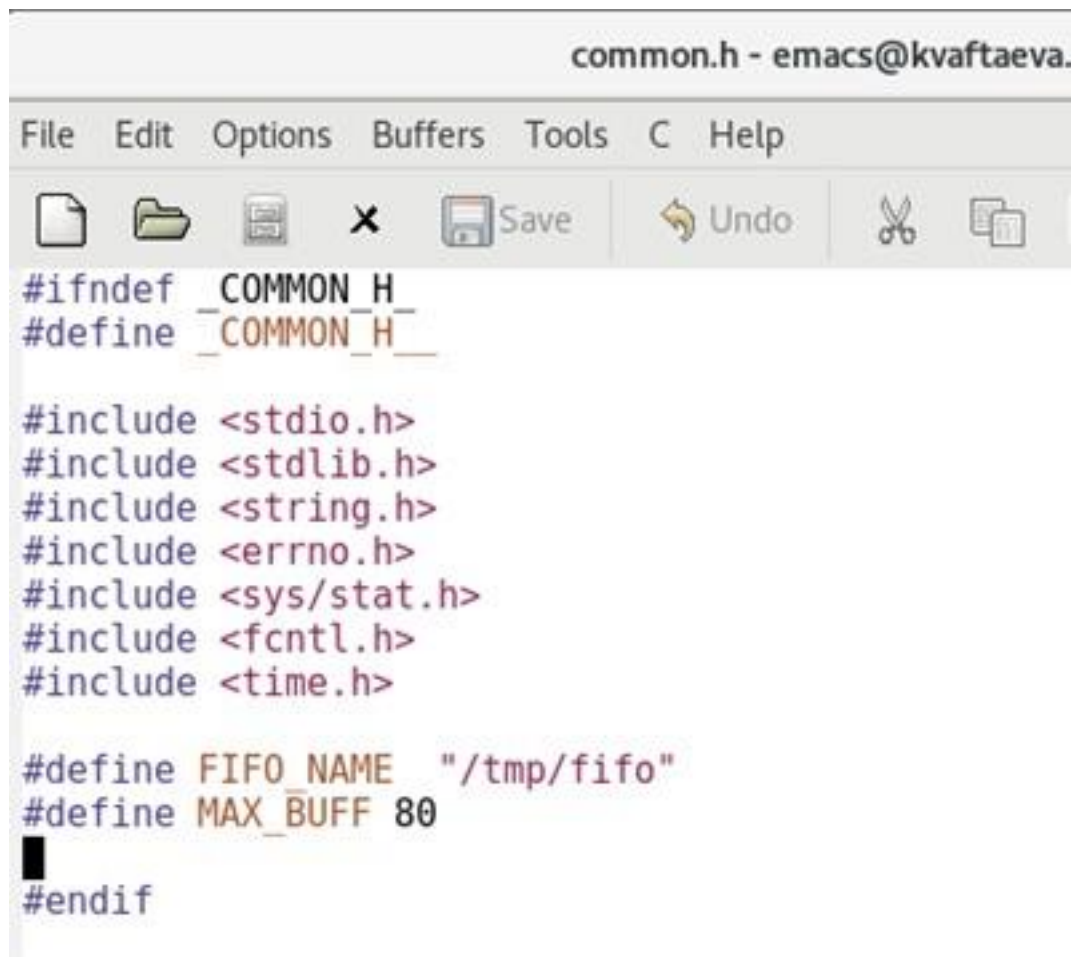
## Выполнение работы:

1. Создала файлы **common.h**, **server.c**, **client.c** и **Makefile** с помощью команды `emacs`, которая создает файлы и сразу открывает их на редактирование в текстовом редакторе **emacs** (Рис.1)

```
[kvaftaeva@kvaftaeva ~]$ emacs common.h
[kvaftaeva@kvaftaeva ~]$ emacs server.c
[kvaftaeva@kvaftaeva ~]$ emacs client.c
[kvaftaeva@kvaftaeva ~]$ emacs Makefile
```

Рис.1 Создание нужных файлов

Файл **common.h** (заголовочный файл со стандартными определениями) (Рис.2)

The image shows a screenshot of the Emacs text editor. The title bar at the top reads "common.h - emacs@kvaftaeva.". Below the title bar is a menu bar with the following items: File, Edit, Options, Buffers, Tools, C, and Help. Under the menu bar is a toolbar with icons for file operations: a new file icon, a folder icon, a save icon, a close icon, a save icon labeled "Save", an undo icon labeled "Undo", a cut icon, and a copy icon. The main editing area contains the following C preprocessor code:

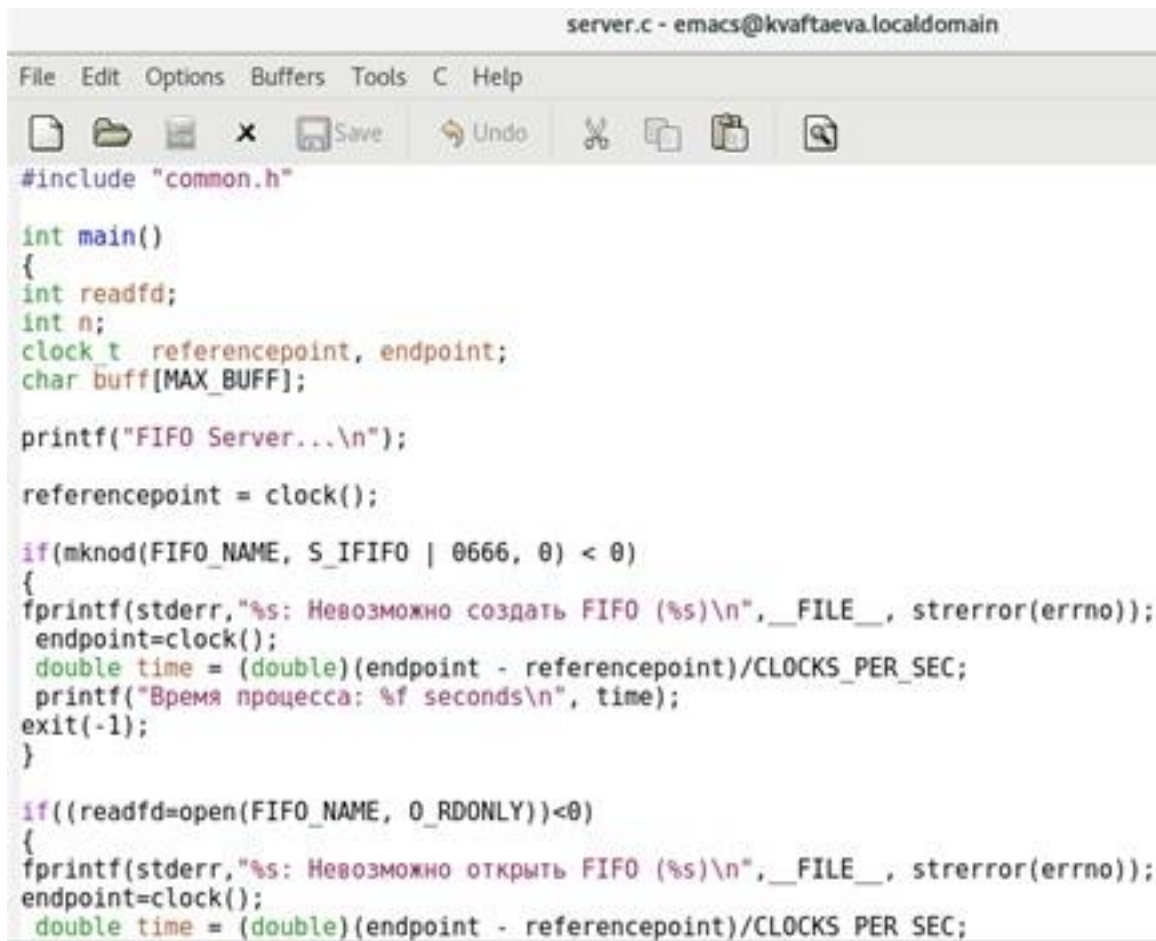
```
#ifndef _COMMON_H_
#define _COMMON_H_

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>

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

Рис.2 Файл *common.h*

Файл **server.c** (реализация сервера) (Рис.3-5)



```
server.c - emacs@kvaftaeva.localdomain
File Edit Options Buffers Tools C Help
Save Undo
#include "common.h"

int main()
{
    int readfd;
    int n;
    clock_t referencepoint, endpoint;
    char buff[MAX_BUFF];

    printf("FIFO Server...\n");

    referencepoint = clock();

    if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)
    {
        fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n", __FILE__, strerror(errno));
        endpoint=clock();
        double time = (double)(endpoint - referencepoint)/CLOCKS_PER_SEC;
        printf("Время процесса: %f seconds\n", time);
        exit(-1);
    }

    if((readfd=open(FIFO_NAME, O_RDONLY))<0)
    {
        fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n", __FILE__, strerror(errno));
        endpoint=clock();
        double time = (double)(endpoint - referencepoint)/CLOCKS_PER_SEC;
```

Рис.3 Файл server.c часть 1

```

    printf("Время процесса: %f seconds\n", time);
    exit(-2);
}

while((n=read(readfd, buff, MAX_BUFF))>0)
{
    if(write(1, buff, n)!=n)
    {
        fprintf(stderr, "%s: Ошибка вывода (%s)\n", __FILE__, strerror(errno));
        endpoint=clock();
        double time = (double)(endpoint - referencepoint)/CLOCKS_PER_SEC;
        printf("Время процесса: %f seconds\n", time);
        exit(-3);
    }
}

close(readfd);

if (unlink (FIFO_NAME)<0)
{
    fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n", __FILE__, strerror (errno));
    endpoint=clock();
    double time = (double)(endpoint - referencepoint)/CLOCKS_PER_SEC;
    printf("Время процесса: %f seconds\n", time);
    exit(-4);
}

```

Рис.4 Файл server.c часть 2

```

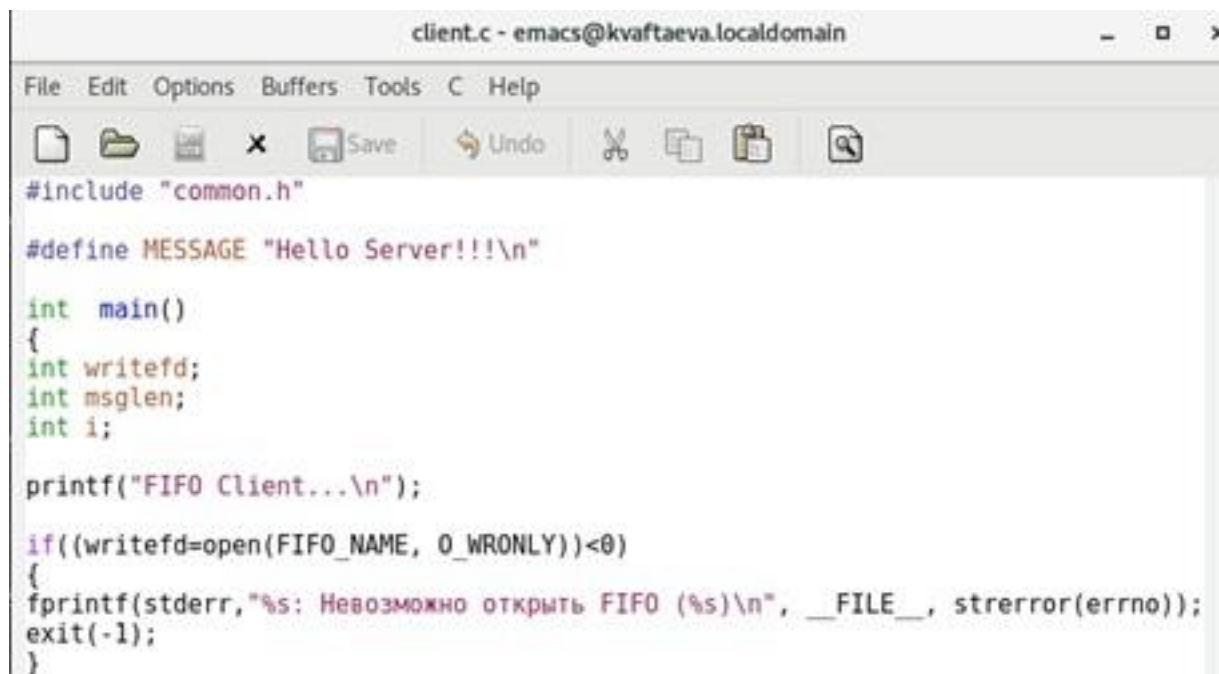
    endpoint=clock();
    double time = (double)(endpoint - referencepoint)/CLOCKS_PER_SEC;
    printf("Время процесса: %f seconds\n", time);

    exit (0);
}

```

Рис.5 Файл server.c часть 3

Файл **client.c** (реализация клиента) (Рис.6-7)



```
client.c - emacs@kvaftaeva.localdomain
File Edit Options Buffers Tools C Help
[Icons] Save Undo [Icons]
#include "common.h"

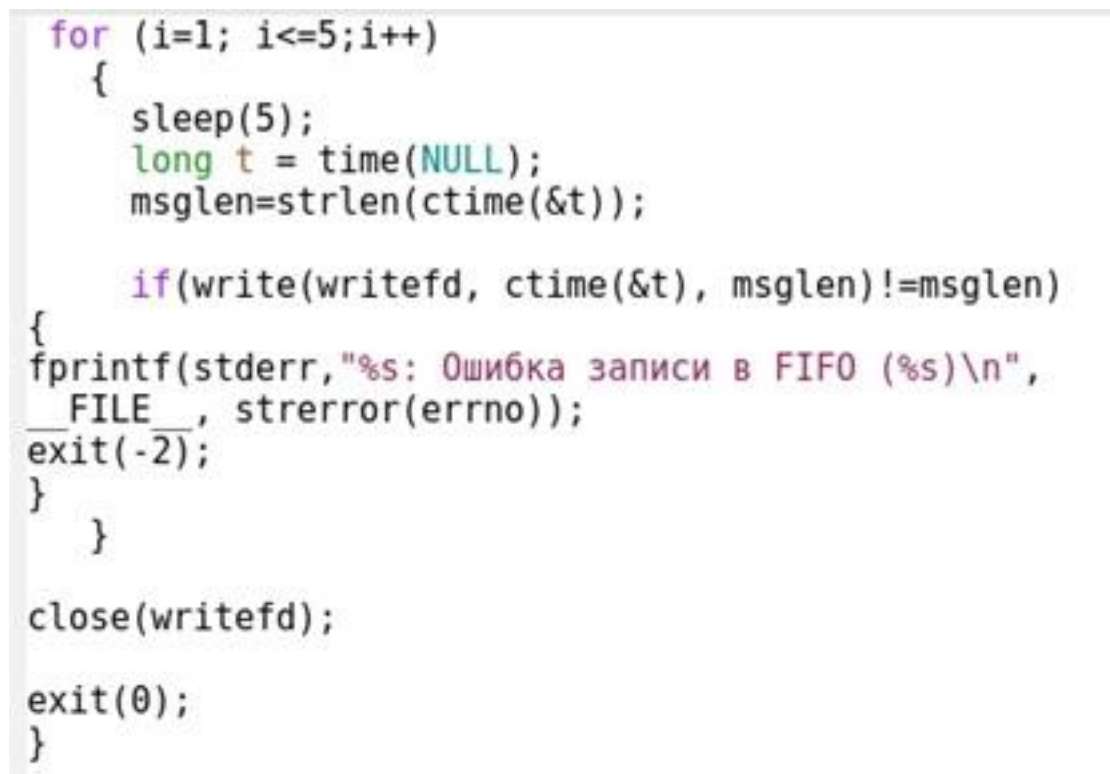
#define MESSAGE "Hello Server!!!\n"

int main()
{
    int writefd;
    int msglen;
    int i;

    printf("FIFO Client...\n");

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

Рис.6 Файл *client.c* часть 1



```
for (i=1; i<=5;i++)
{
    sleep(5);
    long t = time(NULL);
    msglen=strlen(ctime(&t));

    if(write(writefd, ctime(&t), msglen)!=msglen)
    {
        fprintf(stderr,"%s: Ошибка записи в FIFO (%s)\n",
            __FILE__, strerror(errno));
        exit(-2);
    }
}

close(writefd);

exit(0);
}
```

Рис.7 Файл *client.c* часть 2

Файл \*\*Файл Makefile\*



Рис.8 Файл Makefile

После написания файлов сохраняла их с помощью комбинации клавиш `C-x C-s`

Изменения, которые были сделаны:

- работает несколько клиентов (у меня 2)
  - добавила функцию `sleep(5)`, с помощью которой вывод текущей даты и времени происходит с интервалом 5 секунд
  - сервер работает не бесконечно и с помощью функции `clock` выводится время работы
2. Скомпилировала файлы с помощью `gcc` (Рис.9)

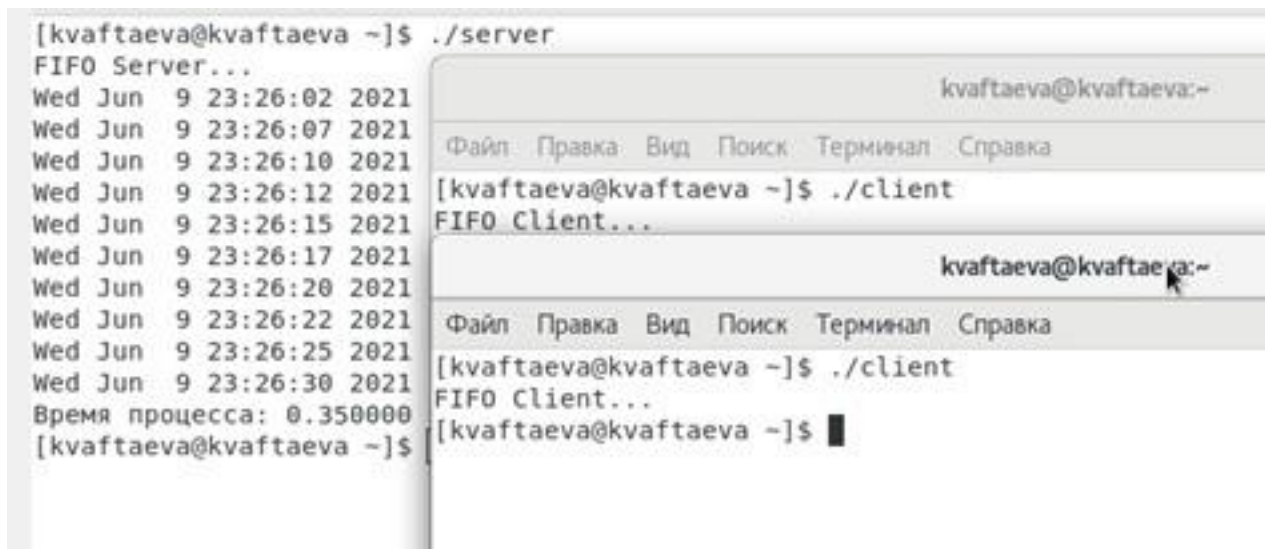
```
[kvaftaeva@kvaftaeva ~]$ gcc -o client client.c
[kvaftaeva@kvaftaeva ~]$ gcc -o server server.c
[kvaftaeva@kvaftaeva ~]$
```

Рис.9 Компиляция

При компиляции во время разработки кода компилятор выводит ошибки, которые были исправлены. Сейчас же компилятор не выводит никаких сообщений, значит код написан верно

3. Проверяю работу программ. Запускаю на одном терминале файл **server.c** с помощью команды `./server`. На двух других запускаю **client.c** с помощью команды `./client` (Рис.10)





```
[kvaftaeva@kvaftaeva ~]$ ./server
FIFO Server...
Wed Jun  9 23:26:02 2021
Wed Jun  9 23:26:07 2021
Wed Jun  9 23:26:10 2021
Wed Jun  9 23:26:12 2021
Wed Jun  9 23:26:15 2021
Wed Jun  9 23:26:17 2021
Wed Jun  9 23:26:20 2021
Wed Jun  9 23:26:22 2021
Wed Jun  9 23:26:25 2021
Wed Jun  9 23:26:30 2021
Время процесса: 0.350000
[kvaftaeva@kvaftaeva ~]$
```

```
[kvaftaeva@kvaftaeva ~]$ ./client
FIFO Client...
Wed Jun  9 23:26:02 2021
Wed Jun  9 23:26:07 2021
Wed Jun  9 23:26:10 2021
Wed Jun  9 23:26:12 2021
Wed Jun  9 23:26:15 2021
Wed Jun  9 23:26:17 2021
Wed Jun  9 23:26:20 2021
Wed Jun  9 23:26:22 2021
Wed Jun  9 23:26:25 2021
Wed Jun  9 23:26:30 2021
[kvaftaeva@kvaftaeva ~]$
```

Рис.10 Работа программы

Видим, что оба клиента выводят текущую дату и время по 5 раз каждый, с интервалом 5 секунд. Время работы программы складывается из суммы интервалов + времени, затраченном на запуск клиентов

---

## Контрольные вопросы: [2]

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. Да, командой `pipe`.

3. Да, командой `$ mkfifo имя_файла`.

4. `int read(int pipe_fd, void *area, int cnt);`

`int write(int pipe_fd, void *area, int cnt);`

Первый аргумент этих вызовов - *дескриптор канала*, второй - *указатель на область памяти*, с которой происходит обмен, третий - *количество байт*. Оба вызова возвращают число переданных байт (или -1 - при ошибке).

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

Первый параметр — имя файла, идентифицирующего канал, второй параметр маска прав доступа к файлу. Вызов функции `mkfifo()` создаёт файл канала (с именем, заданным макросом `FIFO_NAME`): `mkfifo(FIFO_NAME, 0600)`.



6. При чтении меньшего числа байтов, чем находится в канале, 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. 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. 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.
8. В общем случае возможна *многонаправленная работа процессов с каналом*, т.е. возможна ситуация, когда с одним и тем же каналом взаимодействуют два и более процесса, и каждый из взаимодействующих каналов пишет и читает информацию в канал. Но традиционной схемой организации работы с каналом является *однонаправленная организация*, когда канал связывает два, в большинстве случаев, или несколько взаимодействующих процесса, каждый из которых может либо читать, либо писать в канал.
9. Функция записывает **length** байтов из буфера **buffer** в файл, определенный дескриптором файла `fd`. Эта операция чисто 'двоичная' и без буферизации. Реализуется как непосредственный вызов `DOS`. С помощью функции `write` мы посылаем сообщение клиенту или серверу.
10. Функция, транслирующая код ошибки, который обычно хранится в глобальной переменной **errno**, в сообщение об ошибке, понятном человеку. Ошибки эти возникают при вызове функций стандартных Си-библиотек. 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.

---

## Заключение:

Теоретический материал изучен и пригодится в дальнейшей работе. Все цели и задачи выполнены.

## Вывод:

Я приобрела практические навыки работы с **именованными каналами**

---

### Библиографический список:

[1]: Каналы FIFO

[2]: Программирование для Линукс. Профессиональный подход