

# **отчёта по лабораторной работе 15**

Ндри Ив Алла Ролан

# Содержание

0.1	Цель работы . . . . .	4
0.2	Выполнение работы. . . . .	4
0.3	Вывод: . . . . .	8
0.4	Ответы на контрольные вопросы: . . . . .	8

## List of Tables

# List of Figures

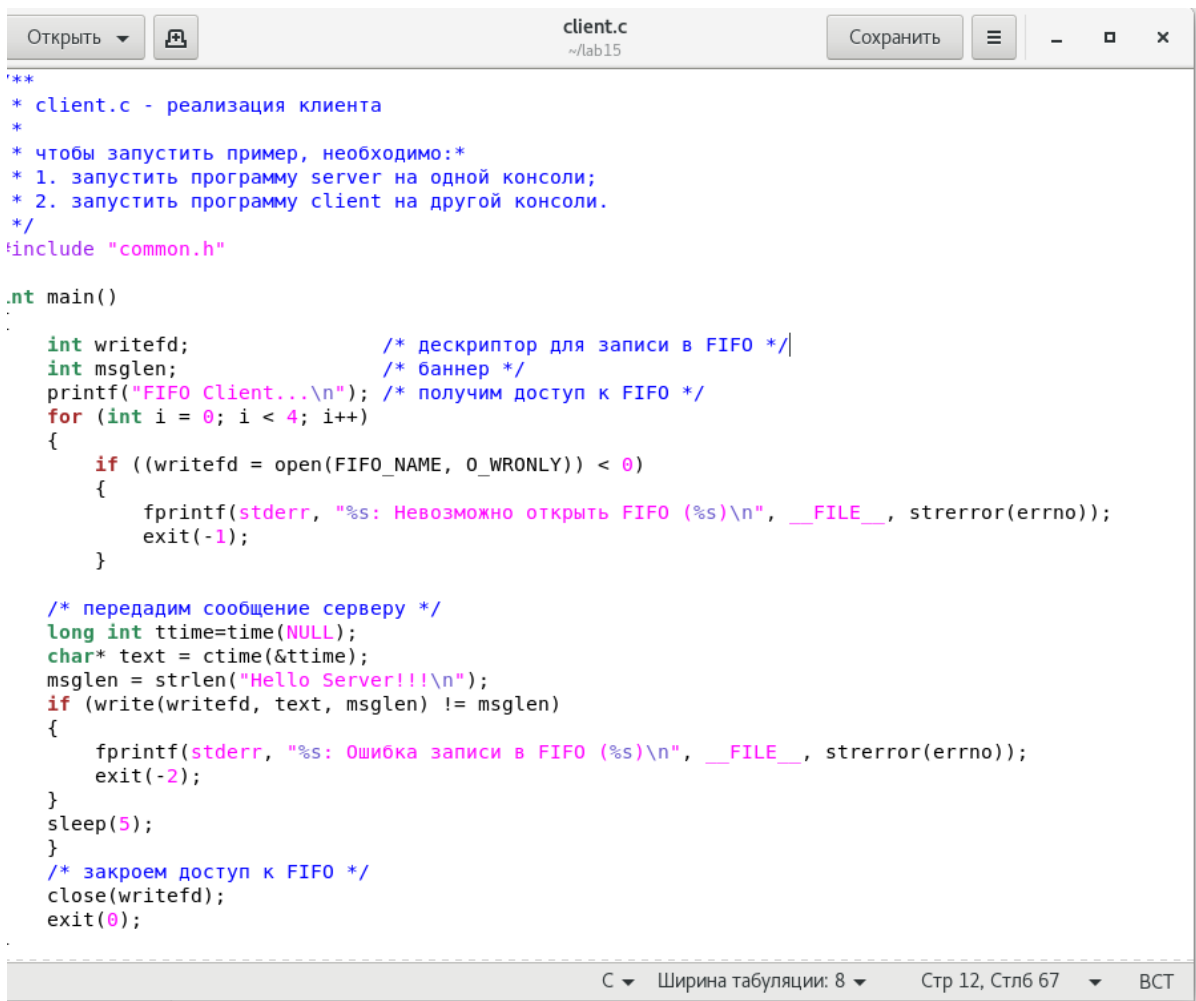
## 0.1 Цель работы

Приобретение практических навыков работы с именованными каналами.

## 0.2 Выполнение работы.

Ход работы: 1. Мы изучили и написали приведенные в тексте задания к лабораторной работе программы `common.h`, `server.c` и `client.c`

(рис. ??)



```
client.c
~/lab15

Открыть Сохранить

/**
 * 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"

int main()
{
    int writefd; /* дескриптор для записи в FIFO */
    int msglen; /* баннер */
    printf("FIFO Client...\n"); /* получим доступ к FIFO */
    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);
        }

        /* передадим сообщение серверу */
        long int ttime=time(NULL);
        char* text = ctime(&ttime);
        msglen = strlen("Hello Server!!!\n");
        if (write(writefd, text, msglen) != msglen)
        {
            fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n", __FILE__, strerror(errno));
            exit(-2);
        }
        sleep(5);
    }
    /* закроем доступ к FIFO */
    close(writefd);
    exit(0);
}
```

С Ширина табуляции: 8 Стр 12, Стлб 67 ВСТ

{ # fig:001 width=70%} (рис. ??)



The screenshot shows a code editor window with the title bar "common.h" and the path "~/lab15". The window contains the following C code:

```
/**
 * 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>

//added
#include <unistd.h>
#include <time.h>

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

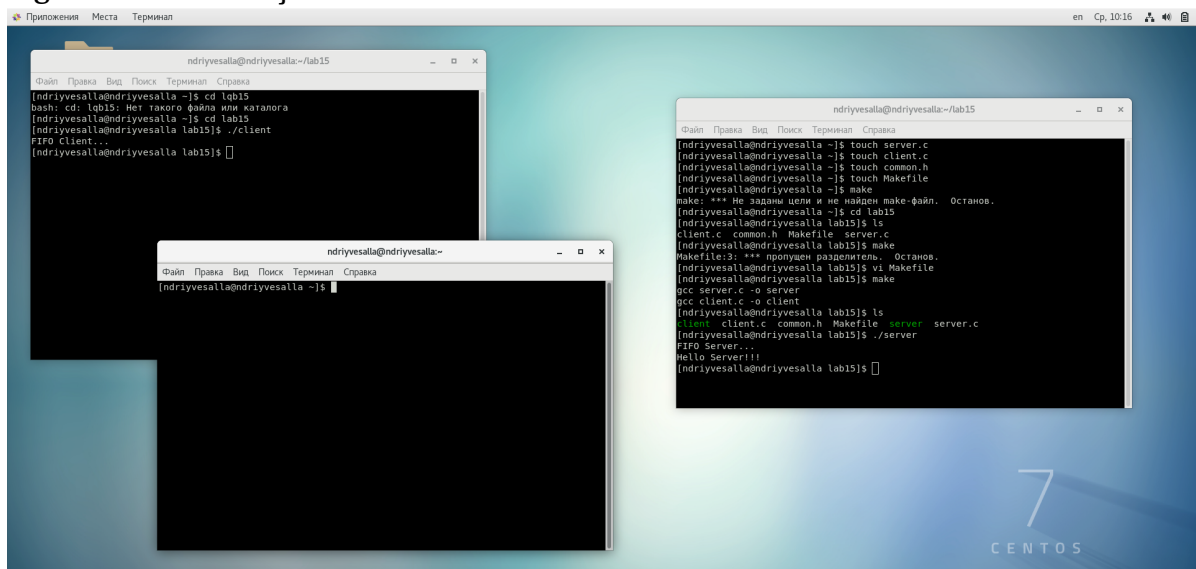
{ # fig:002 width=70%}

(рис. ??)

```
server.c
~/lab15

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);
} /* откроем FIFO на чтение */
if ((readfd = open(FIFO_NAME, O_RDONLY | O_NONBLOCK)) < 0)
{
    fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n", __FILE__, strerror(errno));
    exit(-2);
} /* читаем данные из FIFO и выводим на экран */
clock_t start = time(NULL);
// Проверка на работу в 30 секунд/
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 */ /* удалим FIFO из системы */
    if (unlink(FIFO_NAME) < 0)
    {
        fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n", __FILE__, strerror(errno));
        exit(-4);
    }
    exit(0);
}
```

{ # fig:003 width=70%}



{ # fig:004 width=70%}

Как видно сервер откликается через 5 секунд от каждого нового запуска клиента, а через 30 секунд программа завершается с выводом общего количества времени и интервал от последнего отклика клиента до завершения сервера

Если сервер завершит работу не закрыв канал, то при следующем запуске будет выведено следующее (errno)

## 0.3 Вывод:

Написали FIFO канал на C (server/client)

## 0.4 Ответы на 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).
2. Возможно ли создание неименованного канала из командной строки? Неименованные каналы создаются вызовом `pipe()` и предоставляют возможность только однонаправленной (односторонней) передачи данных:

```
#include <unistd.h> int fd[2]; pipe(fd);
```

```
/* возвращает 0 в случае успешного завершения, -1 - в случае ошибки;*/
```

Функция возвращает два файловых дескриптора: `fd[0]` и `fd[1]`, причем первый открыт для чтения, а второй – для записи.

Возможно создание неименованного канала из командной строки только с созданием временного именованного канала.

3. Возможно ли создание именованного канала из командной строки?

Да, с помощью `mkfifo` или, использованной с давних времен, `mknod`



4. Опишите функцию языка C, создающую неименованный канал. `#include <unistd.h>`

```
int fd[2]; pipe(fd);
```

```
/* возвращает 0 в случае успешного завершения, -1 - в случае ошибки;*/
```

Функция возвращает два файловых дескриптора: `fd[0]` и `fd[1]`, причем первый открыт для чтения, а второй – для записи.

5. Опишите функцию языка C, создающую именованный канал. `#include <sys/types.h>`

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

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

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=EP1PE`) (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?

Да, если введено достаточное количество байтов для `buff`, при этом порции процессов не перемешиваются, иначе, при большем вводе байтов, чем указано, возможна ошибка.

#### 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 записывает `length` байтов из буфера `buffer` в файл, определенный дескриптором файла `fd`. Эта операция чисто ‘двоичная’ и без буферизации. При единице возвращает действительное число байтов.

Функция `write` возвращает число действительно записанных в файл байтов или -1 при ошибке, устанавливая при этом `errno`.

#### 10. Опишите функцию `strerror`

Используется, чтоб получить `pointer` к ошибке в строке буфера и возврата `errno`, выводя сообщения с указанием `errno`. Чаще используется `strerror_s` (secured — защищенный), нежели `strerror` из-за безопасности и утечки данных.