

# **Отчёт по лабораторной работе №14**

**Именованные каналы**

Желдакова Виктория Алексеевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Теоретическое введение</b>	<b>6</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
<b>4</b>	<b>Выводы</b>	<b>12</b>
<b>5</b>	<b>Контрольные вопросы</b>	<b>13</b>

## Список иллюстраций

3.1	Содержимое server.c . . . . .	8
3.2	Содержимое client.c . . . . .	9
3.3	Содержимое client2.c . . . . .	10
3.4	Содержимое common.h . . . . .	10
3.5	Содержимое Makefile . . . . .	11
3.6	Работа программы . . . . .	11

## Список таблиц

# **1 Цель работы**

Приобретение практических навыков работы с именованными каналами.

## 2 Теоретическое введение

Канал - это средство связи стандартного вывода одного процесса со стандартным вводом другого. Каналы - это старейший из инструментов ИРС, существующий приблизительно со времени появления самых ранних версий оперативной системы UNIX. Они предоставляют метод односторонних коммуникаций (отсюда термин half-duplex) между процессами.

Эта особенность широко используется даже в командной строке UNIX (в shell-е).

Именованные каналы во многом работают так же, как и обычные каналы, но все же имеют несколько заметных отличий.

Именованные каналы существуют в виде специального файла устройства в файловой системе. Процессы различного происхождения могут разделять данные через такой канал. Именованный канал остается в файловой системе для дальнейшего использования и после того, как весь ввод/вывод сделан.

## 3 Выполнение лабораторной работы

Изучили приведённые в тексте программы `server.c` и `client.c`. Взяв данные примеры за образец, написали аналогичные программы, (рис. 3.1, рис. 3.2 и рис. 3.3) внося следующие изменения:

1. Работает не 1 клиент, а несколько.
2. Клиенты передают текущее время с некоторой периодичностью.
3. Сервер работает не бесконечно, а прекращает работу через некоторое время.



```
#include "common.h"

int
main()
{
    int readfd;
    int n;
    char buff[MAX_BUFF];
    printf("FIFO Server...\n");

    if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)
    {
        fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n",
            __FILE__, strerror(errno));
        exit(-1);
    }

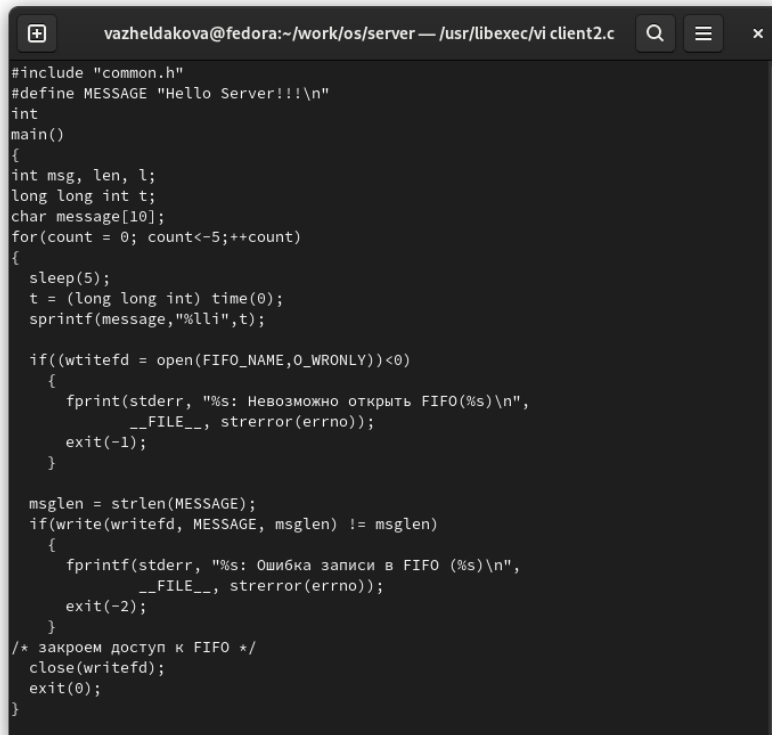
    if((_readfd = open(FIFO_NAME, O_RDONLY)) < 0)
    {
        fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
            __FILE__, strerror(errno));
        exit(-2);
    }

    clock_t now=time(NULL), start=time(NULL);
    while(now-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));
            }
        }
        now=time(NULL);
    }
    printf("server timeout, %li - second passed\n", (now-start));
    close(readfd);
    if(unlink(FIFO_NAME) < 0)
    {
        fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n",
            __FILE__, strerror(errno));
        exit(-4);
    }
    exit(0);
}
```

Рис. 3.1: Содержимое server.c







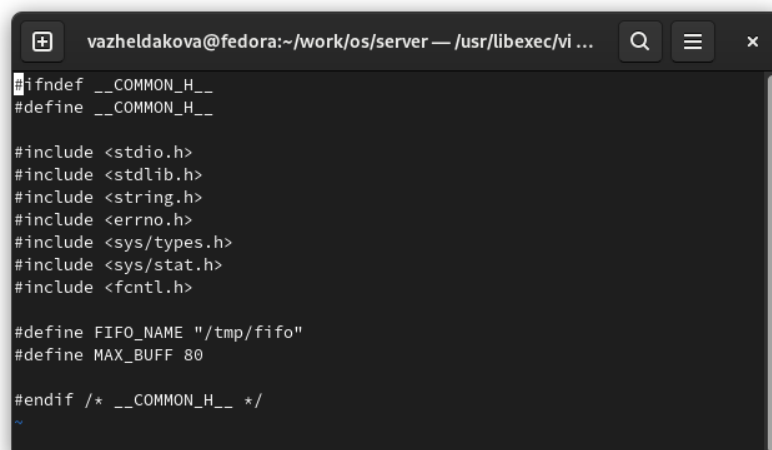
```
#include "common.h"
#define MESSAGE "Hello Server!!!\n"
int
main()
{
    int msg, len, l;
    long long int t;
    char message[10];
    for(count = 0; count<-5;++count)
    {
        sleep(5);
        t = (long long int) time(0);
        sprintf(message,"%lli",t);

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

        msglen = strlen(MESSAGE);
        if(write(writefd, MESSAGE, msglen) != msglen)
        {
            fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
                __FILE__, strerror(errno));
            exit(-2);
        }
    }
    /* закроем доступ к FIFO */
    close(writefd);
    exit(0);
}
```

Рис. 3.3: Содержимое client2.c

Скопировали содержимое файлов common.h и Makefile из лабораторной работы (рис. 3.4 и рис. 3.5)



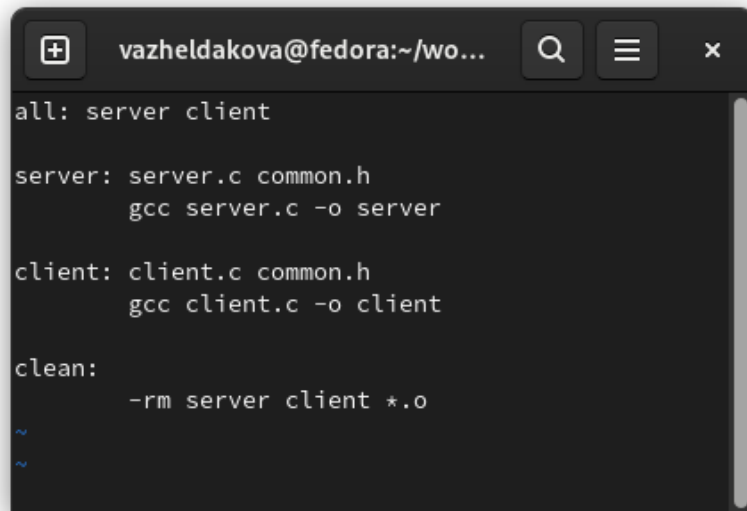
```
#ifndef __COMMON_H__
#define __COMMON_H__

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

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

#endif /* __COMMON_H__ */
```

Рис. 3.4: Содержимое common.h

A terminal window titled 'vazheldakova@fedora:~/wo...' showing the contents of a Makefile. The file defines targets for 'all', 'server', 'client', and 'clean'.

```
all: server client

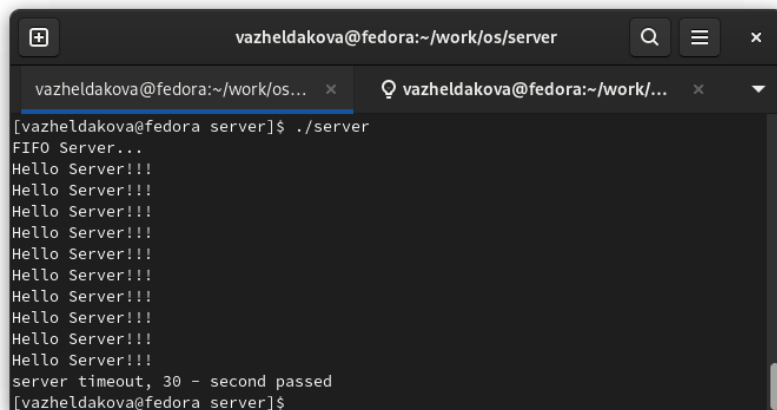
server: server.c common.h
    gcc server.c -o server

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

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

Рис. 3.5: Содержимое Makefile

Запустили make, server в одном окне и client в другом (рис. 3.6)

A terminal window titled 'vazheldakova@fedora:~/work/os/server' showing the execution of the server program. The output shows 'FIFO Server...' followed by ten 'Hello Server!!!' messages and a timeout message.

```
[vazheldakova@fedora server]$ ./server
FIFO Server...
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
server timeout, 30 - second passed
[vazheldakova@fedora server]$
```

Рис. 3.6: Работа программы

Сообщения передалась с периодичностью в несколько секунд, а по истечении 30 секунд - канал закрылся, и вывелось сообщение об этом.

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

При завершении программы каналы автоматически закрываются.

## **4 Выводы**

Приобрели практических навыков работы с именованными каналами.

## 5 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?

Создание неименованного канала из командной строки возможно командой `pipe`.

3. Возможно ли создание именованного канала из командной строки?

Создание именованного канала из командной строки возможно с помощью `mkfifo`.

4. Опишите функцию языка C, создающую неименованный канал.

Функция языка C, создающая неименованный канал: `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. Опишите функцию языка C, создающую именованный канал.

Функция языка C, создающая именованный канал: `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. Что будет в случае прочтения из `fifo` меньшего числа байтов, чем находится в канале? Большого числа байтов?

При чтении меньшего числа байтов, возвращается требуемое число байтов, остаток сохраняется для следующих чтений. При чтении большего числа байтов, возвращается доступное число байтов.

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?

Два и более процессов могут читать и записывать в канал.

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 `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`.

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