

# **Отчет по лабораторной работе №14**

**дисциплина: операционные системы**

**Шмаков Максим Павлович**

# Содержание

Цель работы	5
Задание	6
Выполнение лабораторной работы	7
Выводы	12
Контрольные вопросы	13

# Список иллюстраций

0.1. рис. 1 . . . . .	7
0.2. рис. 2 . . . . .	7
0.3. рис. 3 . . . . .	8
0.4. рис. 4 . . . . .	9
0.5. рис. 5 . . . . .	10
0.6. рис. 6 . . . . .	10
0.7. рис. 7 . . . . .	11

## **Список таблиц**

# **Цель работы**

Приобретение практических навыков работы с именованными каналами

## 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?

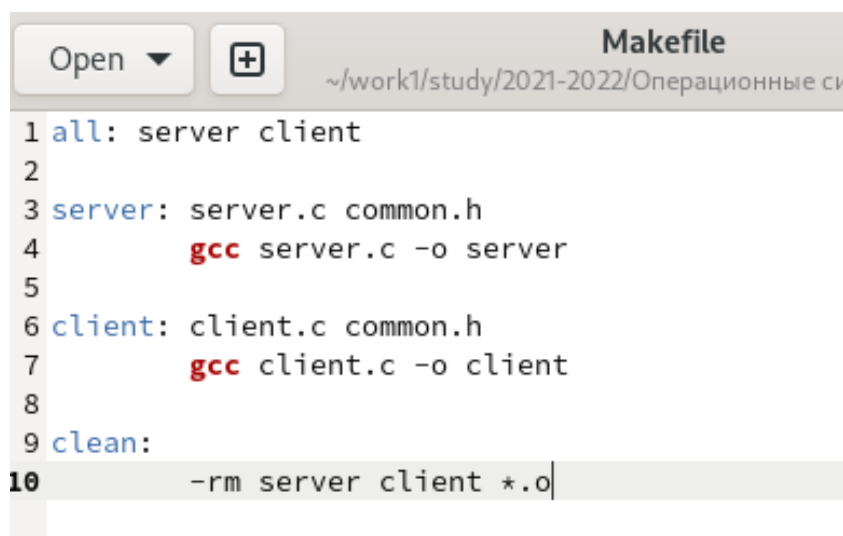
# Выполнение лабораторной работы

Создаю необходимые файлы (рис. [-@fig:001])

```
[mpshmakov@fedora lab14]$ touch common.h server.c client.c Makefile
```

Рис. 0.1.: рис. 1

Далее изменяю коды этих программ. Makefile оставил таким, каким он был. (рис. [-@fig:002])



The screenshot shows a text editor window titled "Makefile" with a path of "~/work1/study/2021-2022/Операционные си". The editor contains the following Makefile rules:

```
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. 0.2.: рис. 2

В файл common.h добавил unistd.h и time.h. Они нужны для корректной работы других файлов. (рис. [-@fig:003])

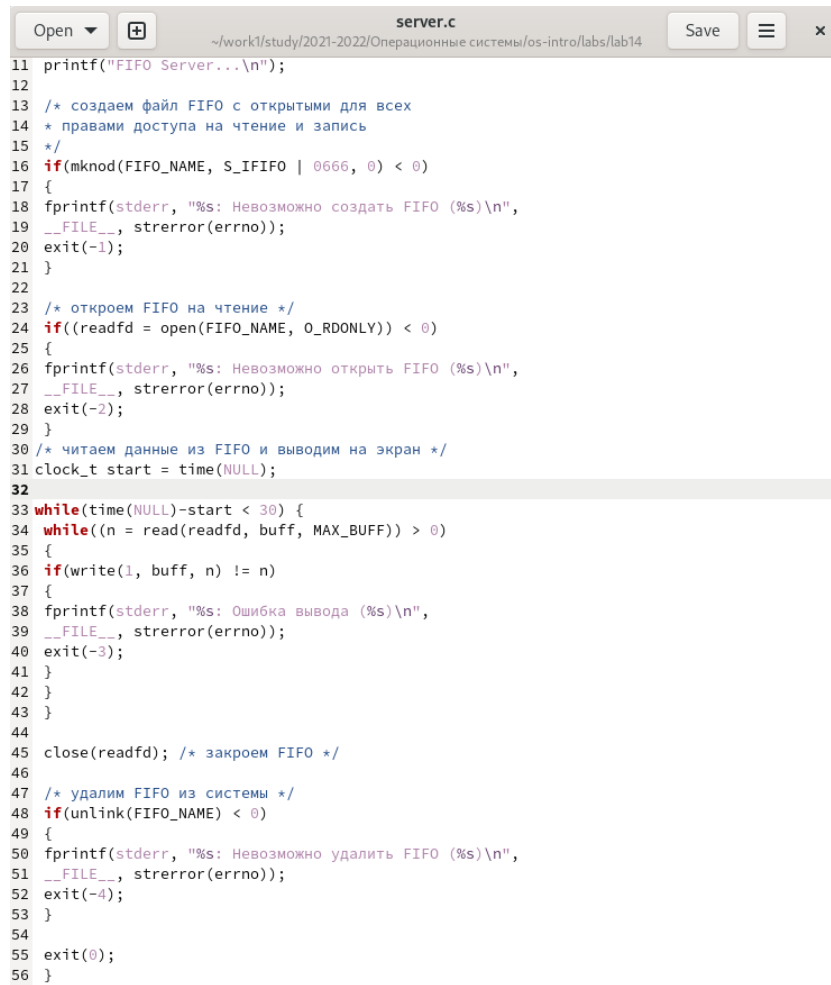


```
1 /*
2  * common.h - заголовочный файл со стандартными определениями
3  */
4
5 #ifndef __COMMON_H__
6 #define __COMMON_H__
7
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <string.h>
11 #include <errno.h>
12 #include <sys/types.h>
13 #include <sys/stat.h>
14 #include <fcntl.h>
15 #include <time.h>
16 #include <unistd.h>
17
18 #define FIFO_NAME "/tmp/fifo"
19 #define MAX_BUFF 80
20
21 #endif /* __COMMON_H__ */
```

Рис. 0.3.: рис. 3

В файл server.c добавил цикл while, который следит, чтобы работа сервера завершилась через 30 секунд. (рис. [-@fig:004])

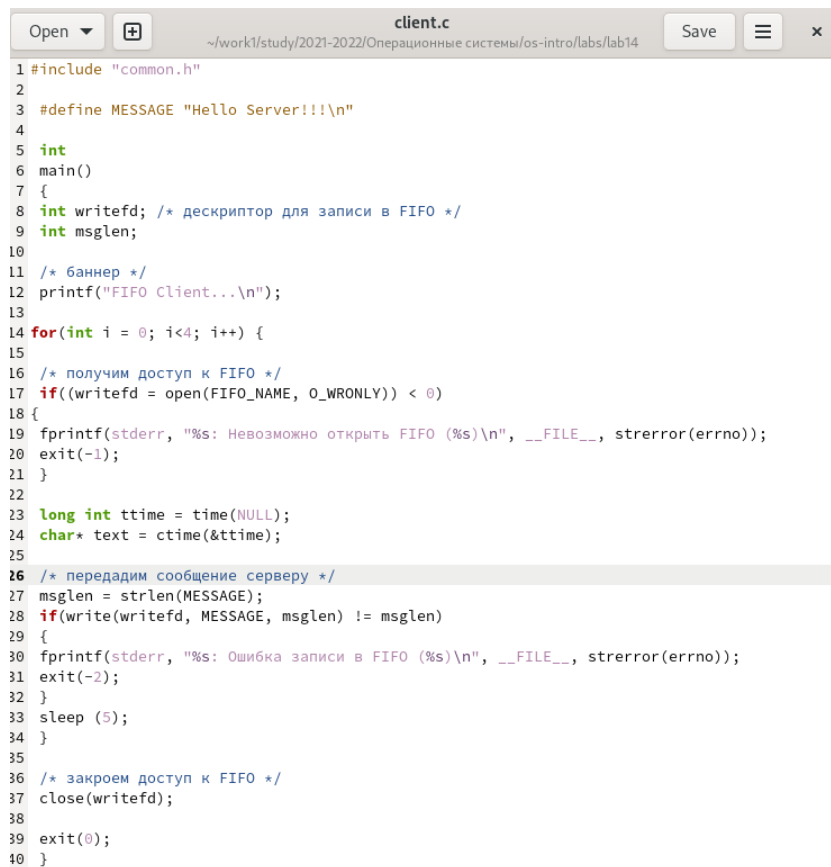




```
11 printf("FIFO Server...\n");
12
13 /* создаем файл FIFO с открытыми для всех
14 * правами доступа на чтение и запись
15 */
16 if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)
17 {
18     fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n",
19             __FILE__, strerror(errno));
20     exit(-1);
21 }
22
23 /* откроем FIFO на чтение */
24 if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
25 {
26     fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
27             __FILE__, strerror(errno));
28     exit(-2);
29 }
30 /* читаем данные из FIFO и выводим на экран */
31 clock_t start = time(NULL);
32
33 while(time(NULL)-start < 30) {
34     while((n = read(readfd, buff, MAX_BUFF)) > 0)
35     {
36         if(write(1, buff, n) != n)
37         {
38             fprintf(stderr, "%s: Ошибка вывода (%s)\n",
39                     __FILE__, strerror(errno));
40             exit(-3);
41         }
42     }
43 }
44
45 close(readfd); /* закроем FIFO */
46
47 /* удалим FIFO из системы */
48 if(unlink(FIFO_NAME) < 0)
49 {
50     fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n",
51             __FILE__, strerror(errno));
52     exit(-4);
53 }
54
55 exit(0);
56 }
```

Рис. 0.4.: рис. 4

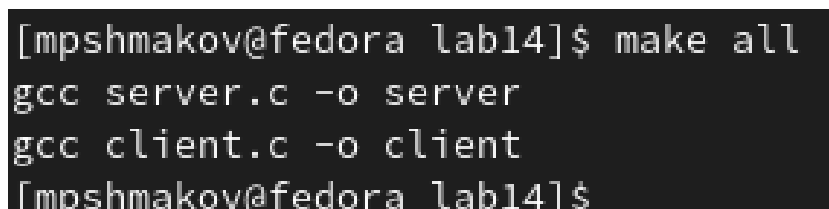
В client.c добавил цикл for, который отвечает за число сообщений (4 сообщения). Также написал команду sleep 5, которая приостанавливает работу клиента на 5 секунд. (рис. [-@fig:005])



```
1 #include "common.h"
2
3 #define MESSAGE "Hello Server!!!\n"
4
5 int
6 main()
7 {
8     int writefd; /* дескриптор для записи в FIFO */
9     int msglen;
10
11     /* баннер */
12     printf("FIFO Client...\n");
13
14     for(int i = 0; i<4; i++) {
15
16         /* получим доступ к FIFO */
17         if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
18         {
19             fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n", __FILE__, strerror(errno));
20             exit(-1);
21         }
22
23         long int ttime = time(NULL);
24         char* text = ctime(&ttime);
25
26         /* передадим сообщение серверу */
27         msglen = strlen(MESSAGE);
28         if(write(writefd, MESSAGE, msglen) != msglen)
29         {
30             fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n", __FILE__, strerror(errno));
31             exit(-2);
32         }
33         sleep (5);
34     }
35
36     /* закроем доступ к FIFO */
37     close(writefd);
38
39     exit(0);
40 }
```

Рис. 0.5.: рис. 5

Компилирую все файлы с помощью Makefile. (рис. [-@fig:006])



```
[mpshmakov@fedora lab14]$ make all
gcc server.c -o server
gcc client.c -o client
[mpshmakov@fedora lab14]$
```

Рис. 0.6.: рис. 6

Далее я проверил работу кода. Открыл 3 консоли. В первой прописал ./server, а на других 2ух ./client. Каждый клиент вывел по 4 сообщения и через 30 секунд сервер прекратил работу. (рис. [-@fig:007])

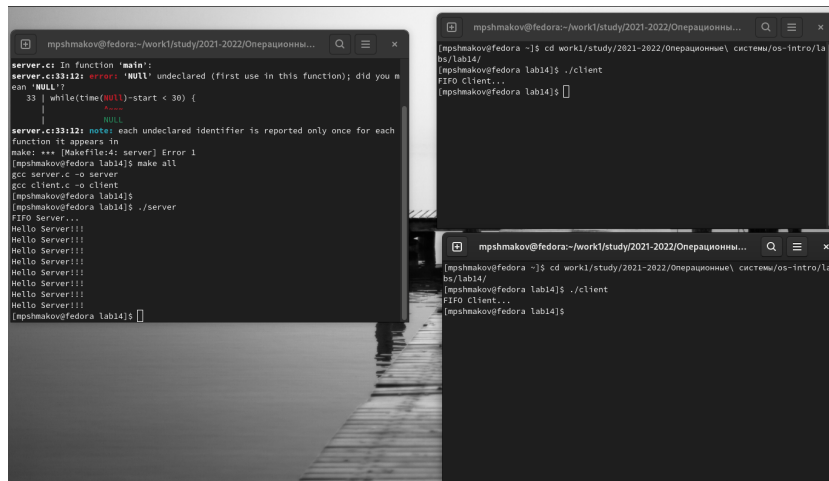


Рис. 0.7.: рис. 7

## **Выводы**

В ходе работы я научился работать с именованными каналами

# 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?

Нет, невозможно.

3. Возможно ли создание именованного канала из командной строки?

Да, возможно. Файлы именованных каналов создаются функцией `mkfifo`.

4. Опишите функцию языка C, создающую неименованный канал.

Неименованный канал создается вызовом `pipe`, который заносит в массив `int [2]` два дескриптора открытых файлов. `fd[0]` – открыт на чтение, `fd[1]` – на запись (вспомните `STDIN == 0`, `STDOUT == 1`). Канал уничтожается, когда будут закрыты все файловые дескрипторы ссылающиеся на него.

5. Опишите функцию языка C, создающую именованный канал.

Объект FIFO в файловой системе создаётся вызовом функции `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=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?

Родитель после записи не может узнать считал ли дочерний процесс данные, а если считал то сколько. Соответственно, если родитель попытается читать из `pipe`, то, вполне вероятно, он считает часть собственных данных, которые станут недоступными для потомка.

Через канал можно передавать данные только между двумя процессами. Один из процессов создает канал, другой открывает его. После этого оба процесса могут передавать данные через канал в одну или обе стороны, используя для этого хорошо знакомые вам функции, предназначенные для работы с файлами, такие как `ReadFile` и `WriteFile`. Заметим, что приложения могут выполнять над каналами `Pipe` синхронные или асинхронные операции, аналогично тому, как это можно делать с файлами. В случае использования асинхронных операций необходимо отдельно побеспокоиться об организации синхронизации.

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.

Возвращаемое значение.

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:

ЗНАЧЕНИЕ ЕГО СМЫСЛ

`EACCES` файл открыт для чтения или закрыт для записи

`EBADF` неверный `handle`-р файла

`ENOSPC` на устройстве нет свободного места

1 в программе `server.c` означает идентификатор стандартного потока вывода.

10. Опишите функцию `strerror`.

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