

# **Лабораторная работа №14**

**Именованные каналы**

СИССЕ МОХАМЕД ЛАМИН;НММбд-01-22

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>6</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
<b>5</b>	<b>Выводы</b>	<b>10</b>
<b>6</b>	<b>Контрольные вопросы</b>	<b>11</b>
	<b>Список литературы</b>	<b>13</b>

## Список иллюстраций

4.1	Текст программы . . . . .	7
4.2	Текст программы . . . . .	8
4.3	Текст программы . . . . .	8
4.4	Текст программы . . . . .	9
4.5	Компиляция . . . . .	9
4.6	Результат . . . . .	9

# **1 Цель работы**

Приобретение практических навыков работы с именованными каналами.

## 2 Задание

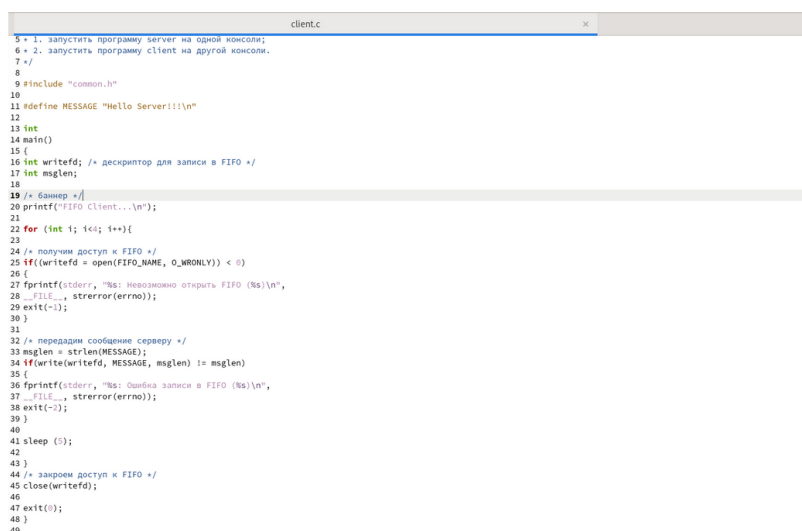
Изучите приведённые в тексте программы `server.c` и `client.c`. Взяв данные примеры за образец, напишите аналогичные программы, внося следующие изменения: 1. Работает не 1 клиент, а несколько (например, два). 2. Клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд). 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 Теоретическое введение

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: общепонимание (именованные каналы, сигналы), System V Interface Definition (SVID — разделяемая память, очередь сообщений, семафоры) и 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)`.  
[Prog: bash?]

## 4 Выполнение лабораторной работы

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. (рис. [4.1])



```
client.c
5 * 1. запустить программу server на одной консоли;
6 * 2. запустить программу client на другой консоли.
7 */
8
9 #include "common.h"
10
11 #define MESSAGE "Hello Server!!!"
12
13 int
14 main()
15 {
16     int writefd; /* дескриптор для записи в FIFO */
17     int msglen;
18
19     /* БANNER */
20     printf("FIFO Client...\n");
21
22     for (int i; i <= 10; i++){
23
24         /* получим доступ к FIFO */
25         if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
26         {
27             fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
28                 __FILE__, strerror(errno));
29             exit(-1);
30         }
31
32         /* передаем сообщение серверу */
33         msglen = strlen(MESSAGE);
34         if(write(writefd, MESSAGE, msglen) != msglen)
35         {
36             fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
37                 __FILE__, strerror(errno));
38             exit(-1);
39         }
40
41         sleep (5);
42     }
43
44     /* закроем доступ к FIFO */
45     close(writefd);
46
47     exit(0);
48 }
49
```

Рис. 4.1: Текст программы




```

Открыть client2.c
1 #include "common.h"
2 #include <time.h>
3 #define MESSAGE "Hello Server!!!"
4
5 int
6 main()
7 {
8     int writefd; /* дескриптор для записи в FIFO */
9     int msglen;
10    long int ttime;
11
12    for(int i=0; i<15; i++)
13    {
14        ttime=time(NULL);
15        printf("time:&ttime");
16        /* баннер */
17        printf("FIFO Client...\n");
18
19        /* получим доступ к FIFO */
20        if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
21        {
22            printf(stderr, "%s: невозможно открыть FIFO (%s)\n",
23                __FILE__, strerror(errno));
24            exit(-1);
25        }
26
27        /* передали сообщение серверу */
28        msglen = strlen(MESSAGE);
29        if(write(writefd, MESSAGE, msglen) != msglen)
30        {
31            printf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
32                __FILE__, strerror(errno));
33            exit(-1);
34        }
35        sleep(1);
36    }
37    /* закроем доступ к FIFO */
38    close(writefd);
39    exit(0);
40 }

```

Рис. 4.2: Текст программы

- Сервер работает не бесконечно, а прекращает работу через некоторое время (напри- мер, 30 сек). Используйте функцию clock() для определения времени работы сервера. Что будет в случае, если сервер завершит работу, не закрыв канал? (рис. [4.3])



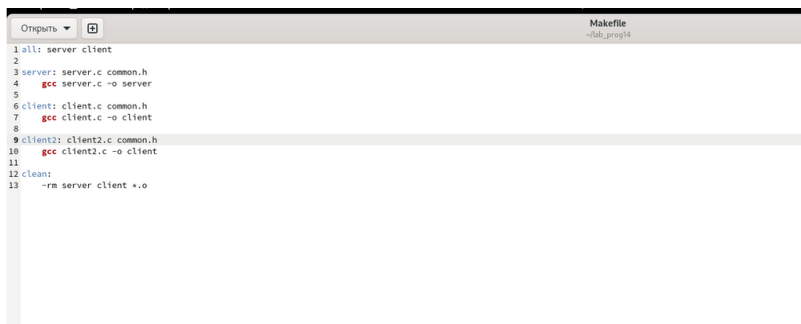
```

server.c
1 /*
2  * server.c - реализация сервера
3  *
4  * чтобы запустить пример, необходимо:
5  * 1. запустить программу server на одной консоли;
6  * 2. запустить программу client на другой консоли.
7  */
8
9 #include "common.h"
10
11 int
12 main()
13 {
14     int readfd; /* дескриптор для чтения из FIFO */
15     int n;
16     char buff[MAX_BUFF]; /* буфер для чтения данных из FIFO */
17
18     /* баннер */
19     printf("FIFO Server...\n");
20
21     /* создаем файл FIFO с открытыми для всех
22      * правами доступа на чтение и запись */
23
24     if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)
25     {
26         printf(stderr, "%s: невозможно создать FIFO (%s)\n",
27             __FILE__, strerror(errno));
28         exit(-1);
29     }
30
31     /* откроем FIFO на чтение */
32     if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
33     {
34         printf(stderr, "%s: невозможно открыть FIFO (%s)\n",
35             __FILE__, strerror(errno));
36         exit(-1);
37     }
38
39     clock_t beginning=time(NULL), clock_t now=time(NULL);
40     while (beginning<now)
41     {
42         /* читаем данные из FIFO и выводим на экран */
43         while((n = read(readfd, buff, MAX_BUFF)) > 0)
44         {
45             if(write(1, buff, n) != n)

```

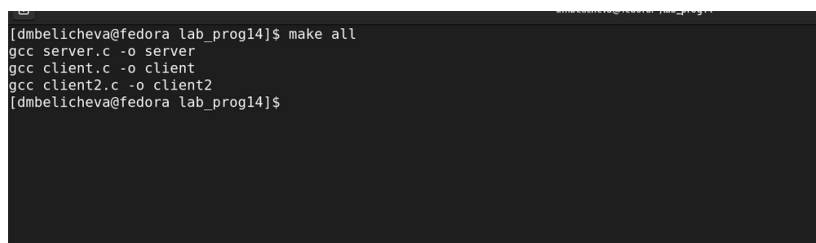
Рис. 4.3: Текст программы





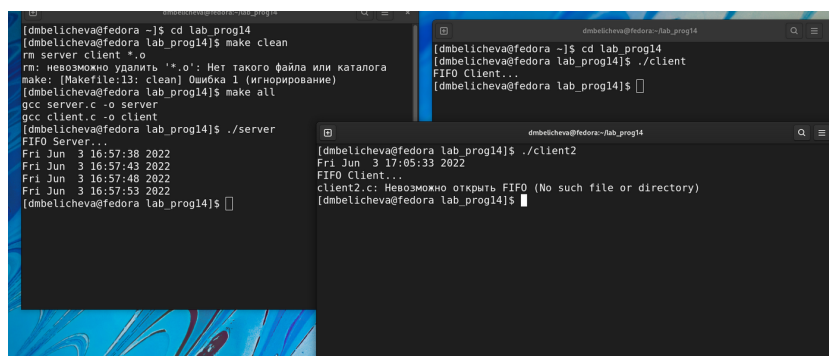
```
1 all: server client
2
3 server: server.c common.h
4     gcc server.c -o server
5
6 client: client.c common.h
7     gcc client.c -o client
8
9 client2: client2.c common.h
10    gcc client2.c -o client2
11
12 clean:
13     -rm server client *.o
```

Рис. 4.4: Текст программы



```
[dmbelicheva@fedora lab_prog14]$ make all
gcc server.c -o server
gcc client.c -o client
gcc client2.c -o client2
[dmbelicheva@fedora lab_prog14]$
```

Рис. 4.5: Компиляция



```
[dmbelicheva@fedora ~]$ cd lab_prog14
[dmbelicheva@fedora lab_prog14]$ make clean
rm server client *.o
rm: невозможно удалить '*.o': Нет такого файла или каталога
make: [Makefile:13: clean] Ошибка 1 (игнорирование)
[dmbelicheva@fedora lab_prog14]$ make all
gcc server.c -o server
gcc client.c -o client
[dmbelicheva@fedora lab_prog14]$ ./server
FIFO Server...
Fri Jun 3 16:57:38 2022
Fri Jun 3 16:57:43 2022
Fri Jun 3 16:57:48 2022
Fri Jun 3 16:57:53 2022
[dmbelicheva@fedora lab_prog14]$

[dmbelicheva@fedora ~]$ cd lab_prog14
[dmbelicheva@fedora lab_prog14]$ ./client
FIFO Client...
[dmbelicheva@fedora lab_prog14]$

[dmbelicheva@fedora ~]$ cd lab_prog14
[dmbelicheva@fedora lab_prog14]$ ./client2
FIFO client...
client2.c: невозможно открыть FIFO (No such file or directory)
[dmbelicheva@fedora lab_prog14]$
```

Рис. 4.6: Результат

## **5 Выводы**

В процессе выполнения лабораторной работы я приобрела практические навыки работы с именованными каналами.

## 6 Контрольные вопросы

1. Именованные каналы отличаются от неименованных наличием идентификатора канала, который представлен как специальный файл (соответственно имя именованного канала — это имя файла).
2. Создание неименованного канала из командной строки возможно командой `pipe`.
3. Создание именованного канала из командной строки возможно с помощью `mkfifo`.
4. 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. 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. При чтении меньшего числа байтов, возвращается требуемое число байтов, остаток сохраняется для следующих чтений. При чтении большего числа

байтов, возвращается доступное число байтов 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=EPipe`) (если процесс не установил обработки сигнала `SIGPIPE`, производится обработка по умолчанию – процесс завершается).

7. Два и более процессов могут читать и записывать в канал.
8. 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`.
9. Строковая функция `strerror` - функция языков C/C++, транслирующая код ошибки, который обычно хранится в глобальной переменной `errno`, в сообщение об ошибке, понятном человеку.

## **Список литературы**