

Лабораторная работа №14

Именованные каналы

Парфенова Елизавета Евгеньевна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	16
5	Контрольные вопросы	17

Список иллюстраций

3.1	Папка с программами	7
3.2	client.c	8
3.3	client2.c	9
3.4	common.h	11
3.5	makefile	12
3.6	server.c	13
3.7	Запуск make	15
3.8	Работа сервера	15

Список таблиц

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 Выполнение лабораторной работы

Я заранее написала аналогичные программы и поместила их в папку lab14.
(рис. 3.1)

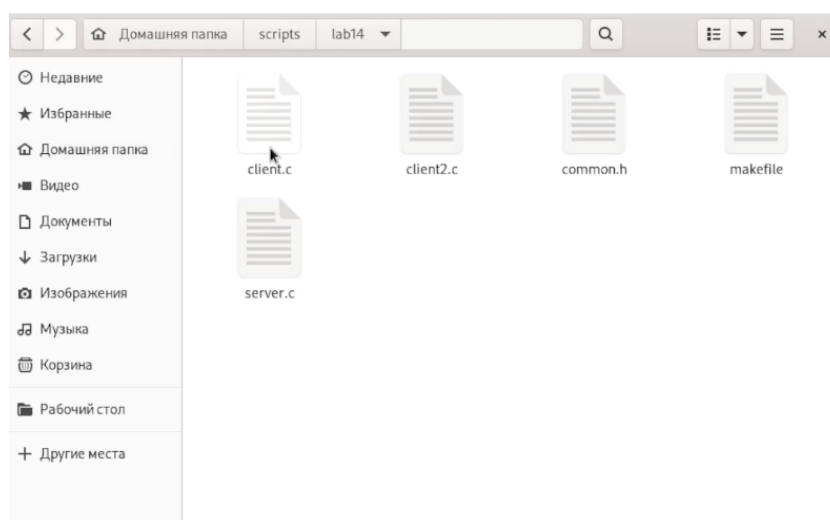


Рис. 3.1: Папка с программами

Пройдемся по каждому из файлов.

1. client.c (рис. 3.2)

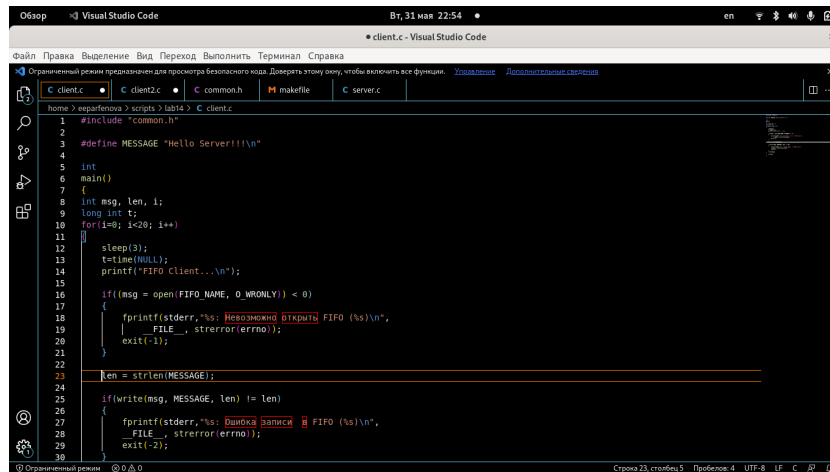


Рис. 3.2: client.c

Листинг:

#include "common.h"

#define MESSAGE "Hello Server!!!"

int

main()

{

int msg, len, i;

long int t;

for(i=0; i<20; i++)

{

sleep(3);

t=time(NULL);

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

if((msg = open(FIFO_NAME, O_WRONLY)) < 0)

{

fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",

__FILE__, strerror(errno));

exit(-1);


```
}
```

```
len = strlen(MESSAGE);
```

```
if(write(msg, MESSAGE, len) != len)
```

```
{
```

```
    fprintf(stderr,"%s: Ошибка записи в FIFO (%s)\n",
```

```
    __FILE__, strerror(errno));
```

```
    exit(-2);
```

```
}
```

```
close(msg);
```

```
    } exit(0); }
```

2. client2.c (рис. 3.3)

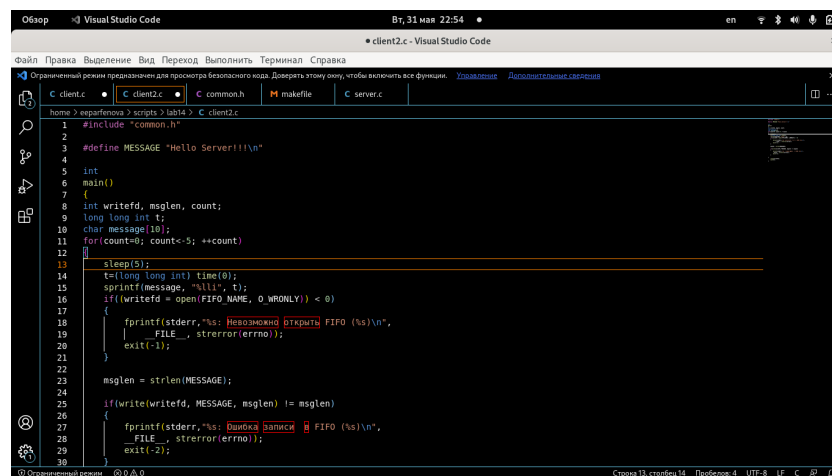


Рис. 3.3: client2.c

Листинг:

```
#include "common.h"
```

```
#define MESSAGE "Hello Server!!!"
```

```
int
```

```

main()
{
    int writefd, msglen, count;
    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);
}

} close(writefd); exit(0); }

```

3. common.h (рис. 3.4)

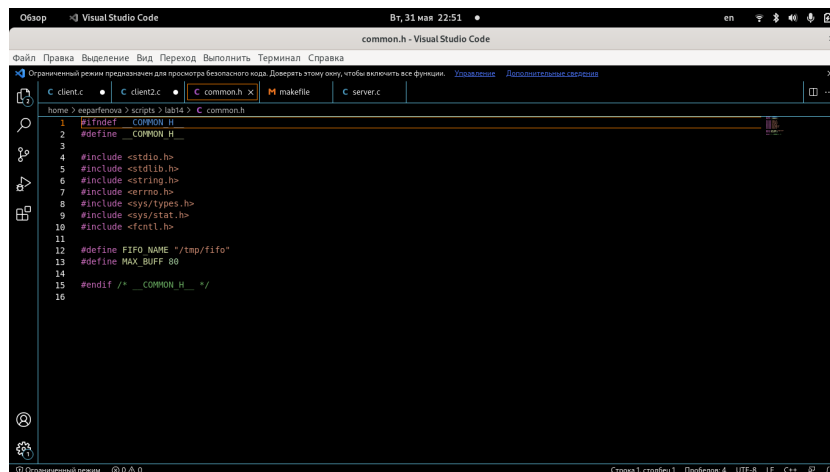


Рис. 3.4: common.h

Листинг:

```
#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 */
```

4. makefile (рис. 3.5)

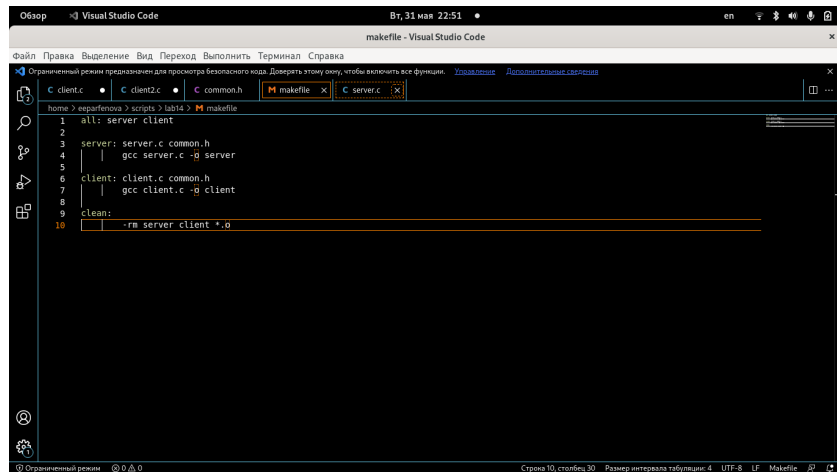


Рис. 3.5: makefile

Листинг:

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

5. server.c (рис. 3.6)

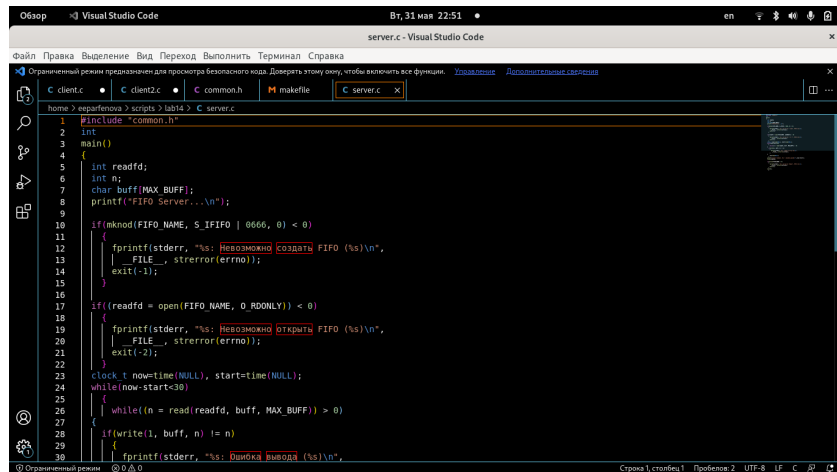


Рис. 3.6: server.c

Листинг:

```

#include "common.h"

int
main()
{
    int readfd;
    int n;
    char buff[MAX_BUFF];
    printf("FIFO Server...");
    if(mknod(FIFO_NAME, S_IFIFO, 0) < 0)
    {
        fprintf(stderr, "%s: Невозможно создать FIFO (%s)",
            __FILE__, strerror(errno));
        exit(-1);
    }

    if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
    {

```

```

    fprintf(stderr, "%s: Невозможно открыть FIFO (%s)",
        __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)",
                __FILE__, strerror(errno));
        }
    }
    now=time(NULL);
}

printf("server timeout, %li - seconds passed",
    (now-start));
close(readfd);
if(unlink(FIFO_NAME) < 0)

{
    fprintf(stderr, "%s: Невозможно удалить FIFO (%s)",
        __FILE__, strerror(errno));
    exit(-4);
}

exit(0);}

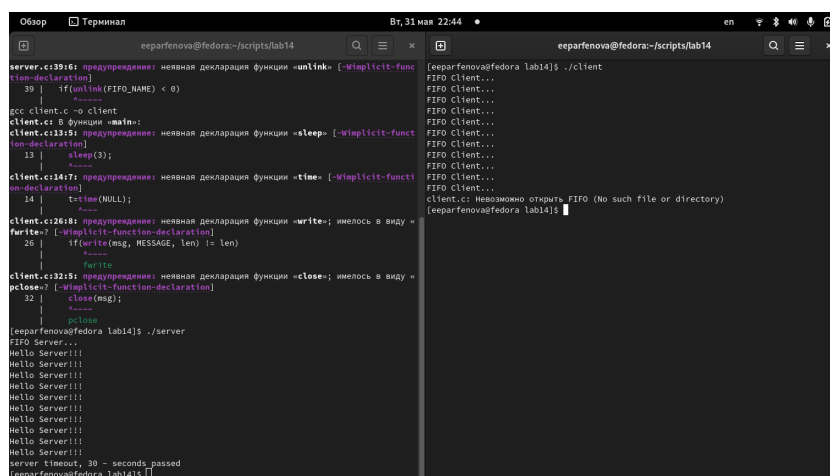
```

Следующим шагом я открыла папку, в которой хранятся все файлы, в терминале и запустила `make` одноименной командой. (рис. 3.7)

```
[eeeparfenova@fedora lab14]$ make
gcc server.c -o server
server.c: В функции «main»:
```

Рис. 3.7: Запуск `make`

Когда все завершилось, в этом же терминале я запустила команду `./server`. Далее снова открыла эту же папку в терминале (нужно было новое окно) и запустила команду `./client`. Необходимо оставить оба терминала работающими. Сервер успешно запустился и закончил работу через 30 секунд. (рис. 3.8)



```
Обзор Терминал
eeeparfenova@fedora:~/scripts/lab14
server.c:39:16: предупреждение: неявная декларация функции «unlink» [-Wimplicit-function-declaration]
   39 |     if(unlink(FIFO_NAME) < 0)
      |                ^
      |                ~~~~~
gcc client.c -o client
client.c:13:19: предупреждение: неявная декларация функции «sleep» [-Wimplicit-function-declaration]
   13 |         sleep(3);
      |         ^~~~~~
client.c:14:17: предупреждение: неявная декларация функции «time» [-Wimplicit-function-declaration]
   14 |         ttime(NULL);
      |         ^~~~~~
client.c:26:18: предупреждение: неявная декларация функции «write»; имелось в виду «fwrite» [-Wimplicit-function-declaration]
   26 |         if(write(msg, MESSAGE, len) != len)
      |         ^~~~~~
client.c:32:15: предупреждение: неявная декларация функции «close»; имелось в виду «fclose» [-Wimplicit-function-declaration]
   32 |         close(msg);
      |         ^~~~~~
[eeeparfenova@fedora lab14]$ ./server
FIFO Server...
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
server timeout, 30 - seconds passed
[eeeparfenova@fedora lab14]$

[eeeparfenova@fedora lab14]$ ./client
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
client.c: Невозможно открыть FIFO (No such file or directory)
[eeeparfenova@fedora lab14]$
```

Рис. 3.8: Работа сервера

4 Выводы

Мы приобрели практические навыки 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. 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.