

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

№ 14

Именованные каналы

Коняева Марина Александровна

Содержание

Цель работы	3
Теоретическое введение	4
Выполнение лабораторной работы	5
Выводы	9
Контрольные вопросы	10

Цель работы

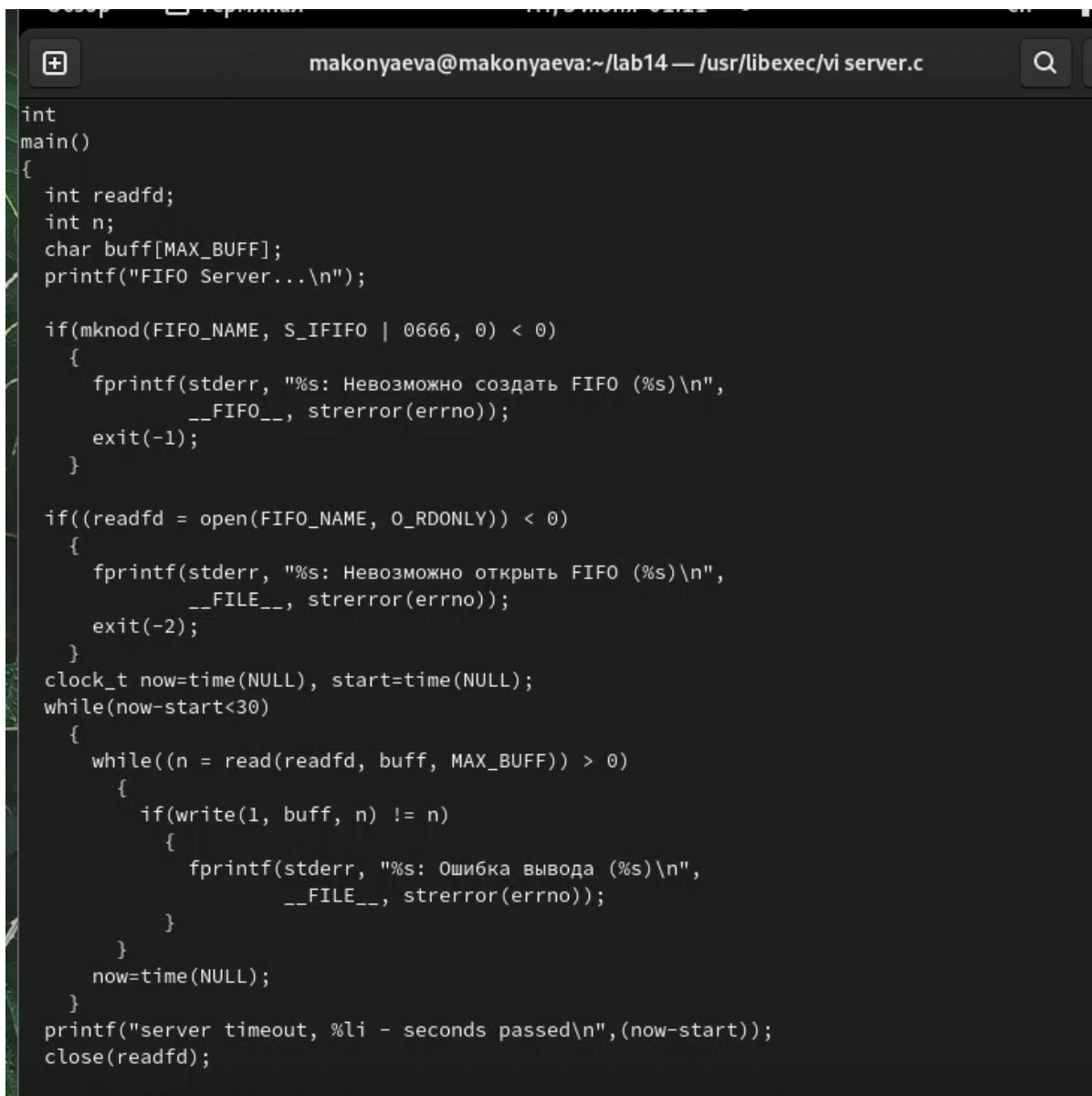
Приобретение практических навыков работы с именованными каналами.

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: общепюниксные (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.

Выполнение лабораторной работы

1. Скрипт 1-4 (изображение 1.1-4)



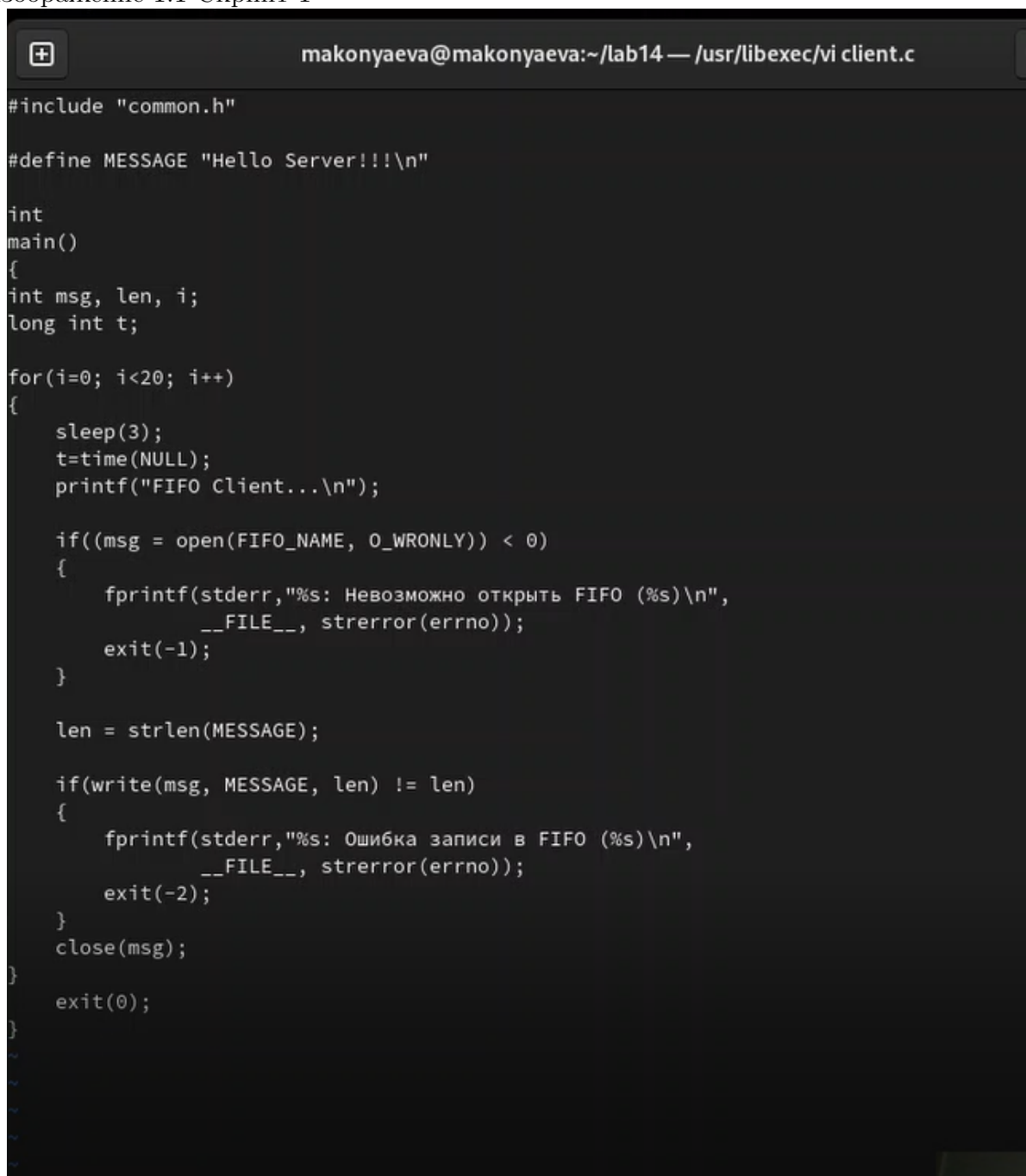
```
makonyaeva@makonyaeva:~/lab14 — /usr/libexec/vi server.c
int
main()
{
    int readfd;
    int n;
    char buff[MAX_BUFF];
    printf("FIFO Server...\n");

    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 - seconds passed\n", (now-start));
    close(readfd);
}
```

Изображение 1.1 Скрипт 1



The image shows a terminal window with a dark background. The title bar at the top reads 'makonyaeva@makonyaeva:~/lab14 — /usr/libexec/vi client.c'. The terminal displays the source code of a C program. The code includes a header file 'common.h', defines a message 'Hello Server!!!\n', and implements a main function. The main function contains a loop that runs 20 times. In each iteration, it sleeps for 3 seconds, gets the current time, prints 'FIFO Client...\n', attempts to open a FIFO file named 'FIFO_NAME' in write-only mode, and if successful, writes the message to it. If the file cannot be opened or the write fails, it prints an error message to stderr and exits with a non-zero status. If the write is successful, it closes the file and continues the loop. After the loop, it prints 'exit(0);' and returns.

```
#include "common.h"

#define MESSAGE "Hello Server!!!\n"

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);
}
```

Изображение 1.2 Скрипт 2

```
#include "common.h"

#define MESSAGE "Hello Server!!!\n"

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);
}
```

Изображение 1.3 Скрипт 3

```
all: server client

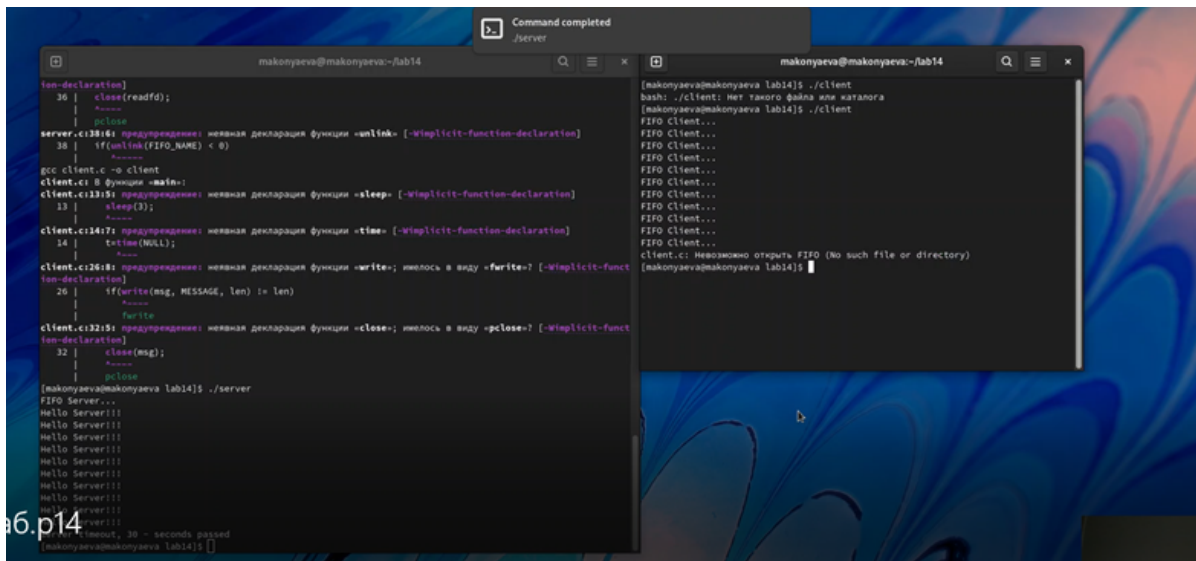
server: server.c common.h
    gcc server.c -o server

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

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

Изображение 1.4 Скрипт 4

2. Запустим в разных консолях (изображение 2.1)



Изображение 2.1 Запустим в разных консолях

Выводы

В ходе данной лабораторной работы приобрели практических навыков работы с именованными каналами, а также ответили на контрольные вопросы.

Контрольные вопросы

1. Именованные каналы отличаются от неименованных наличием имени (странно, не правда ли?), то есть идентификатора канала, потенциально видимого всем процессам системы. Для идентификации именованного канала создается файл специального типа `pipe`.
2. Для создания неименованного канала используется системный вызов `pipe`. Массив из двух целых чисел является выходным параметром этого системного вызова. Если вызов выполнен нормально, то массив содержит два файловых дескриптора: для чтения информации из канала и для записи в него соответственно.
3. Чтобы создать именованный канал, используется команда: `mkfifo` . Она создает файл именованного канала, который можно использовать даже в нескольких сеансах оболочки. Другой способ создать именованный канал FIFO - использовать следующую команду: `mknode p` . Чтобы перенаправить стандартный вывод любой команды другому процессу, используйте символ `>` . Чтобы перенаправить стандартный ввод любой команды, используйте символ `<` .
4. Для создания неименованного канала используется системный вызов `pipe`. Массив из двух целых чисел является выходным параметром этого системного вызова. Если вызов выполнен нормально, то массив содержит два файловых дескриптора: для чтения информации из канала и для записи в него соответственно.

5. Использование пайпов для передачи данных внутри одного процесса – дичайший оверкилл, хуже было бы разве что использовать сокеты. Для передачи данных из потока в поток используйте разделяемую структуру данных (например, `std::queue`), защищая её от одновременного доступа при помощи мьютекса (например, `CRITICAL_SECTION`, если хотите Windows-specific, или лучше `std::mutex`, если ваш компилятор поддерживает стандарт C++11).
6. 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. 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. 3. Если канал пуст и ни один процесс не открыл его на запись, при чтении из канала будет получено 0 байтов. . . . Это означает, что в случае, когда несколько процессов одновременно записывают в канал, порции данных от этих процессов не перемешиваются. 5. 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.
8. Однако если два или более процесса записывают в канал данные одновременно, каждый процесс за один раз может записать максимум `PIPE_BUF` байтов данных. Предположим, процесс (назовем его А) пытается записать X байтов данных в канал, в котором имеется место для Y байтов данных. Если X больше, чем Y, только первые Y байтов данных записываются в канал, и процесс блокируется. Запускается другой процесс (например. В); в это время в канале появляется свободное пространство (благодаря третьему процессу, считывающему данные из канала). Процесс В записывает данные в канал.
9. Функция — это самостоятельная единица программы, которая спроектирована

для реализации конкретной подзадачи. Функция является подпрограммой, которая может содержаться в основной программе, а может быть создана отдельно (в библиотеке). Каждая функция выполняет в программе определенные действия. . . . Если функция не возвращает значения, то тип возвращаемого значения для нее указывается как `void`. При этом операция `return` может быть опущена.

10. Функция `strerror` формирует описание ошибки по коду ошибки указанному в аргументе `errcode` и возвращает указатель на строку, содержащую сформированное описание ошибки.