

Отчёта по лабораторной работе №15

Операционный Систем

Коне Сирики НФИБД-01-20

Содержание

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

List of Tables

List of Figures

3.1	рисунок 1	7
3.2	рисунок 2	8
3.3	рисунок 3	8
3.4	рисунок 4	9
3.5	рисунок 5	9
3.6	рисунок 6	10
3.7	рисунок 7	10
3.8	рисунок 8	11
3.9	рисунок 9	11
3.10	рисунок 10	12
3.11	рисунок 11	12
3.12	рисунок 12	13
3.13	рисунок 13	13

1 Цель работы

приобретение практических навыков работы с именованными каналами.

2 Задание

Именованные каналы

3 Выполнение лабораторной работы

Ход работы: 1. Изучил приведённые в тексте программы server.c и client.c и взяла данные примеры за образец.

common.h:

(рис. 3.1)

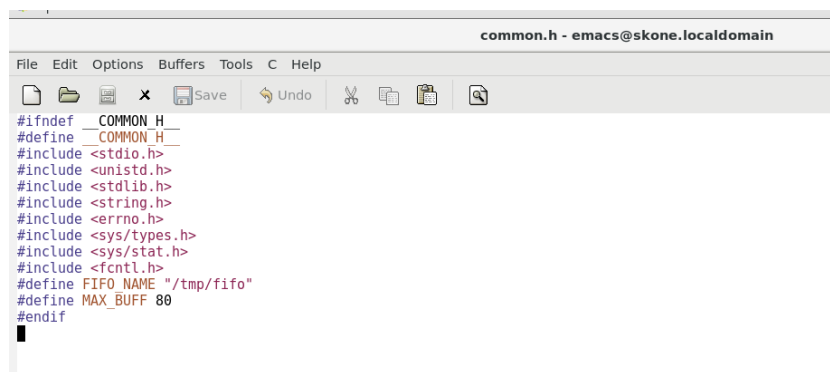
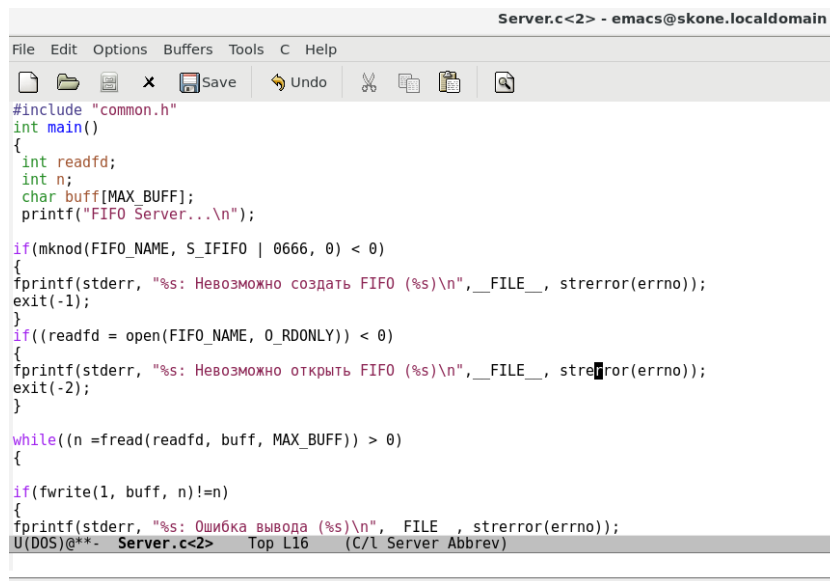


Figure 3.1: рисунок 1

server.c:

(рис. 3.2)



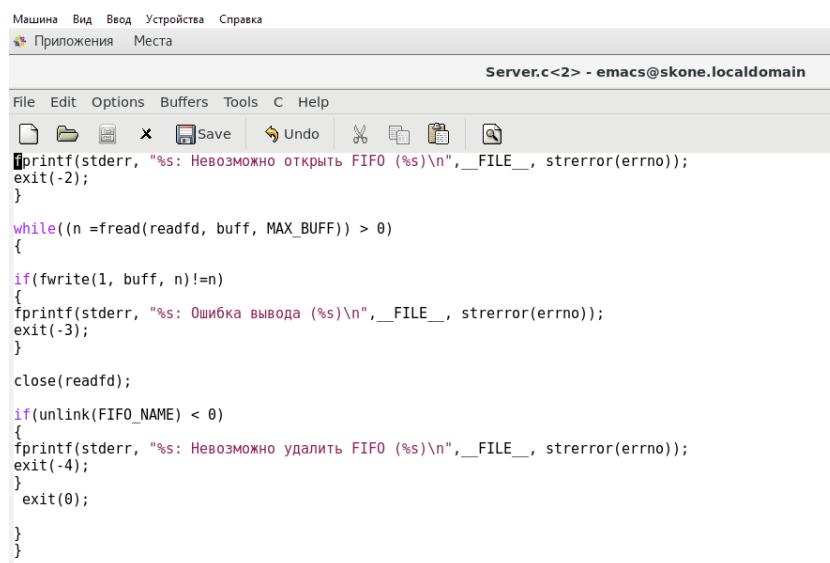
```
Server.c<2> - emacs@skone.localdomain
File Edit Options Buffers Tools C Help
#include "common.h"
int main()
{
    int readfd;
    int n;
    char buff[MAX_BUFF];
    printf("FIFO Server...\n");

    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);
    }

    while((n = fread(readfd, buff, MAX_BUFF)) > 0)
    {
        if(fwrite(1, buff, n) != n)
        {
            fprintf(stderr, "%s: Ошибка вывода (%s)\n", __FILE__, strerror(errno));
            U(DOS)@**~ Server.c<2> Top L16 (C/l Server Abbrev)
        }
    }
}
```

Figure 3.2: рисунок 2

(рис. 3.3)



```
Машинa Вид Ввод Устройство Справка
Приложения Места
Server.c<2> - emacs@skone.localdomain
File Edit Options Buffers Tools C Help
printf(stderr, "%s: Невозможно открыть FIFO (%s)\n", __FILE__, strerror(errno));
exit(-2);
}

while((n = fread(readfd, buff, MAX_BUFF)) > 0)
{
    if(fwrite(1, buff, n) != n)
    {
        fprintf(stderr, "%s: Ошибка вывода (%s)\n", __FILE__, strerror(errno));
        exit(-3);
    }

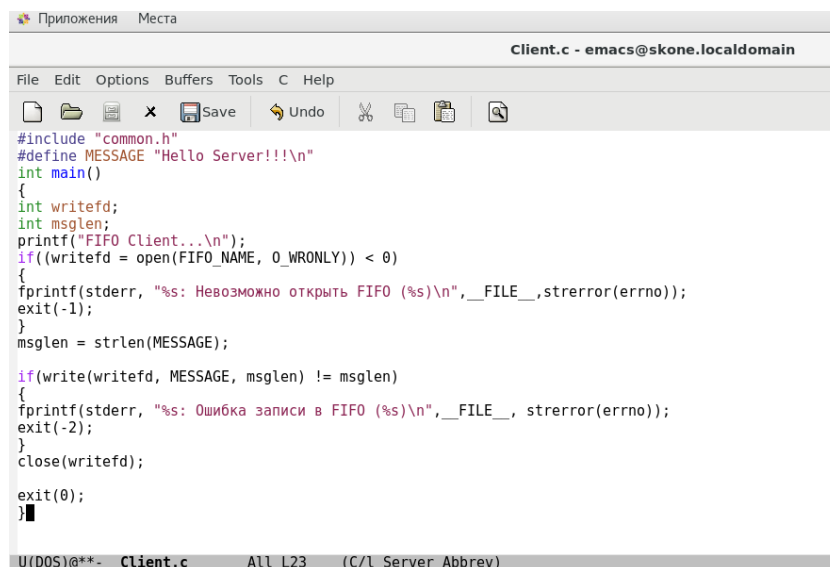
    close(readfd);

    if(unlink(FIFO_NAME) < 0)
    {
        fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n", __FILE__, strerror(errno));
        exit(-4);
    }
    exit(0);
}
}
```

Figure 3.3: рисунок 3

client.c:

(рис. 3.4)



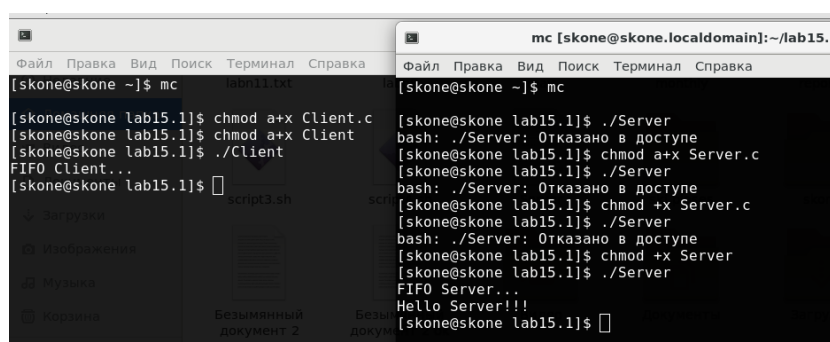
```
#include "common.h"
#define MESSAGE "Hello Server!!!\n"
int main()
{
    int writefd;
    int msglen;
    printf("FIFO Client...\n");
    if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
    {
        fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n", __FILE__, strerror(errno));
        exit(-1);
    }
    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);
}
```

Figure 3.4: рисунок 4

(рис. 3.5)



```
[skone@skone ~]$ mc
[skone@skone lab15.1]$ gcc -o Client Client.c
[skone@skone lab15.1]$ ./Client
FIFO Client...

[skone@skone ~]$ mc
[skone@skone lab15.1]$ gcc -o Server Server.c
[skone@skone lab15.1]$ ./Server
FIFO Server...
Hello Server!!!
```

Figure 3.5: рисунок 5

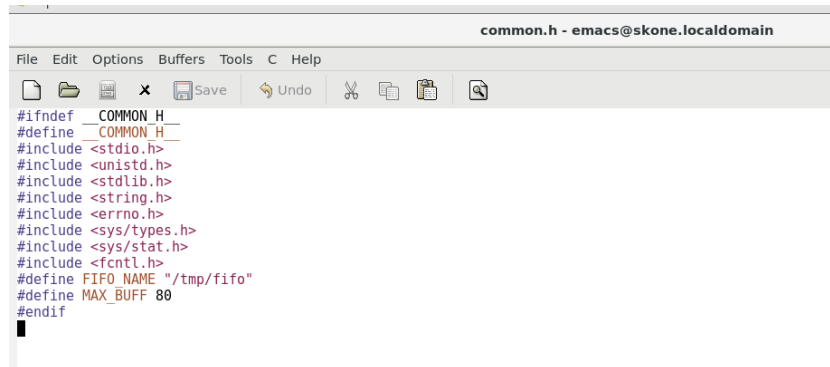
2. Написал аналогичные программы, внося следующие изменения:

- работает не 1 клиент, а несколько (например, два).
- клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд). Использовала функцию `sleep()` для приостановки работы клиента.

3. сервер работает не бесконечно, а прекращает работу через некоторое время (например, 30 сек). Использовала функцию `clock()` для определения времени работы сервера.

common.h:

(рис. 3.6)

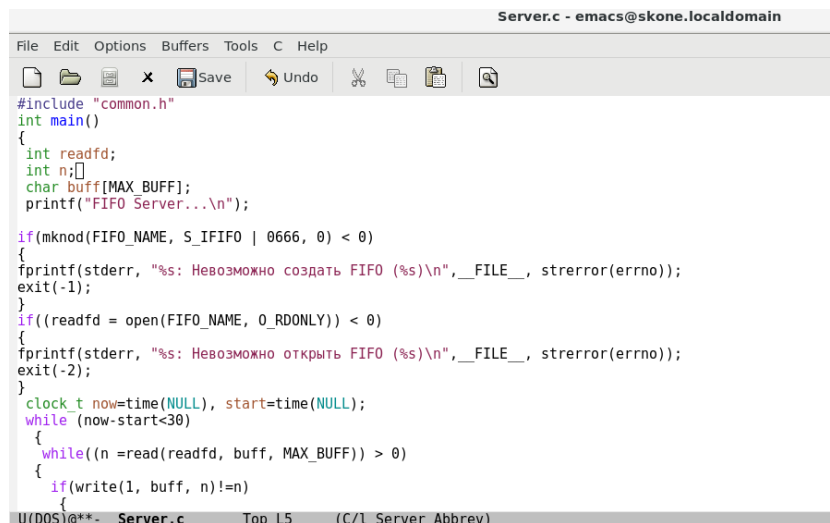


```
common.h - emacs@skone.localdomain
File Edit Options Buffers Tools C Help
[Icons] Save Undo [Icons]
#ifdef COMMON_H
#define COMMON_H
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#define FIFO_NAME "/tmp/fifo"
#define MAX_BUFF 80
#endif
```

Figure 3.6: рисунок 6

server.c:

(рис. 3.7)



```
Server.c - emacs@skone.localdomain
File Edit Options Buffers Tools C Help
[Icons] Save Undo [Icons]
#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)
            {

```

Figure 3.7: рисунок 7

(рис. 3.8)

```

Server.c - emacs@skone.localdomain
File Edit Options Buffers Tools C Help
[Icons] Save Undo [Icons]
clock_t now=time(NULL), start=time(NULL);
while (now-start<30)
{
    while((n=read(readfd, buff, MAX_BUFF)) > 0)
    {
        if(write(l, buff, n)!=n)
        {
            fprintf(stderr, "%s: Ошибка вывода (%s)\n", __FILE__, strerror(errno));
            exit(-3);
        }
    }
    now=time(NULL);
}

printf("\n----\nserver timeout\n%i секунда прошла время истекло!!!\n----\n", now-start);
if(unlink(FIFO_NAME) < 0)
{
    fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n", __FILE__, strerror(errno));
    exit(-4);
}
exit(0);
}

```

Figure 3.8: рисунок 8

client.c:

(рис. 3.9)

```

Client.c - emacs@skone.localdomain
File Edit Options Buffers Tools C Help
[Icons] Save Undo [Icons]
#include "common.h"
#define MESSAGE "Hello Server!!!\n"
int main()
{
    int writefd;
    int msglen;
    printf("FIFO Client...\n");
    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);
}

```

U(DOS)@*- Client.c All L23 (C/l Server Abbrev)

Figure 3.9: рисунок 9

client1.c:

(рис. 3.10)

```
Client1.c - emacs@skone.localdomain
File Edit Options Buffers Tools C Help
[Icons] Save Undo [Icons]
#include "common.h"
#define MESSAGE "Hello Server!!!\n"
int main()
{
    int writefd;
    int msglen;
    char message[10]
    int count;
    long long int T;
    for (count=0; count<=5; ++count){
        sleep(5);
        T=(long long int) time(0);
        sprintf(message, "%lli", T);
        message[9]='\n';
        printf("FIFO Client...\n");
        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));
        }
    }
}
U: @--- Client1.c Top L14 (C/l Server Abbrev)
```

Figure 3.10: рисунок 10

(рис. 3.11)

```
Client1.c - emacs@skone.localdomain
File Edit Options Buffers Tools C Help
[Icons] Save Undo [Icons]
char message[10]
int count;
long long int T;
for (count=0; count<=5; ++count){
    sleep(5);
    T=(long long int) time(0);
    sprintf(message, "%lli", T);
    message[9]='\n';
    printf("FIFO Client...\n");
    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);
}
U: @--- Client1.c 16% L14 (C/l Server Abbrev)
```

Figure 3.11: рисунок 11

(рис. 3.12)

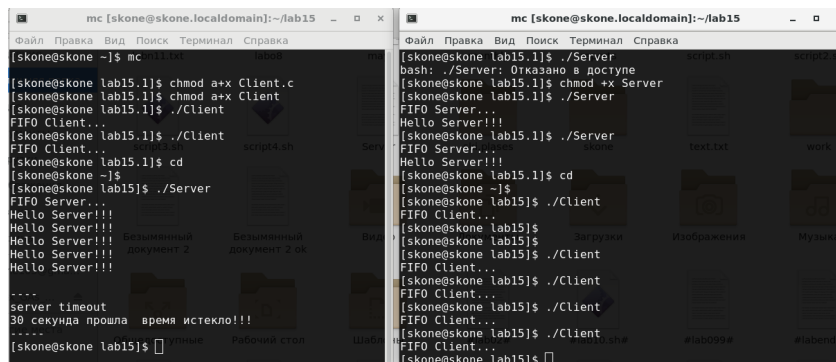


Figure 3.12: рисунок 12

(рис. 3.13)

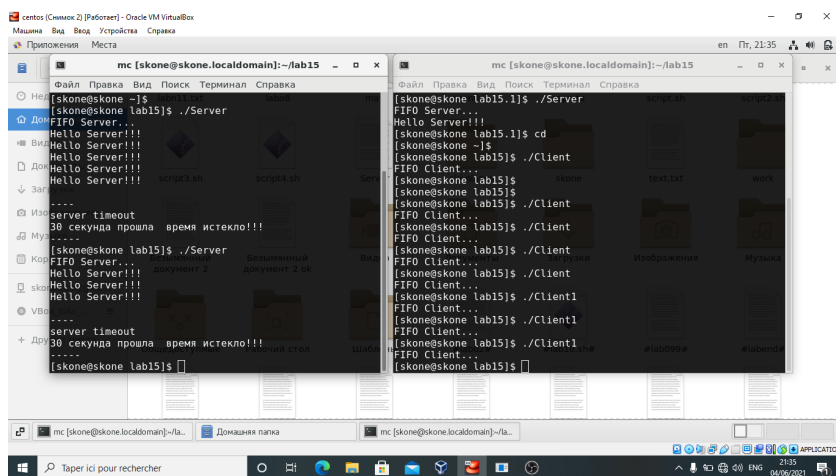


Figure 3.13: рисунок 13

В случае, если сервер завершит работу, не закрыв канал, файл FIFO не удалится, поэтому его в следующий раз создать будет нельзя и вылезет ошибка, следовательно, работать ничего не будет.

Контрольные вопросы:

1. 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. 2.Создание неименованного канала из командной строки невозможно. 3.Создание именованного канала из командной строки возможно.

2. `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).
3. `int mkfifo (const char *pathname, mode_t mode) ; mkfifo(FIFO_NAME, 0600)`
; 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`).
4. При чтении меньшего числа байтов, чем находится в канале, возвращается требуемое число байтов, остаток сохраняется для последующих чтений. При чтении большего числа байтов, чем находится в канале или FIFO возвращается доступное число байтов.
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. Если процесс пытается записать данные в канал, не открытый ни одним процессом на чтение, процессу генерируется сигнал. 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.
6. В общем случае возможна много направленная работа процессов с каналом, т.е. возможна ситуация, когда с одним и тем же каналом взаимодействуют два и более процесса, и каждый из взаимодействующих каналов пишет и читает информацию в канал. Но традиционной схемой организации работы с каналом является однонаправленная организация, когда канал связывает

два, в большинстве случаев, или несколько взаимодействующих процесса, каждый из которых может либо читать, либо писать в канал.

7. Write - Функция записывает length байтов из буфера buffer в файл, определенный дескриптором файла fd. Эта операция чисто 'двоичная' и без буферизации. Реализуется как непосредственный вызов DOS. С помощью функции write мы посылаем сообщение клиенту или серверу.
8. Строковая функция strerror - функция языков C/C++, транслирующая код ошибки, который обычно хранится в глобальной переменной errno, в сообщение об ошибке, понятном человеку. Ошибки эти возникают при вызове функций стандартных Си-библиотек. Возвращенный указатель ссылается на статическую строку с ошибкой, которая не должна быть изменена программой. Дальнейшие вызовы функции strerror перезапишут содержание этой строки. Интерпретированные сообщения об ошибках могут различаться, это зависит от платформы и компилятора.

4 Выводы

Приобрел практические навыки работы с именованными каналами.