

Лабораторная работа №14

Отчёт по лабораторной работе №14

Макарова Анастасия Михайловна

Содержание

Цель работы

Приобретение практических навыков работы с именованными каналами.

Выполнение лабораторной работы

1. Изучим приведённые в тексте программы server.c и client.c. Для написания программ создадим каталог и файлы с помощью команды touch (Рис.1).

```
[ammakarova@10 ~]$ mkdir lab14_progs  
[ammakarova@10 ~]$ cd lab14_progs  
[ammakarova@10 lab14_progs]$ touch server.c client.c common.h Makefile
```

Рис.1

2. Изменим код программы common.h. Добавим стандартные заголовочные файлы, необходимые для работы других файлов. Этот файл предназначен для заголовочных файлов, чтобы не прописывать их в других программах (Рис.2).

```
/*
 * common.h - заголовочный файл со стандартными определениями
 */

#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>
#include <unistd.h>
#include <time.h>

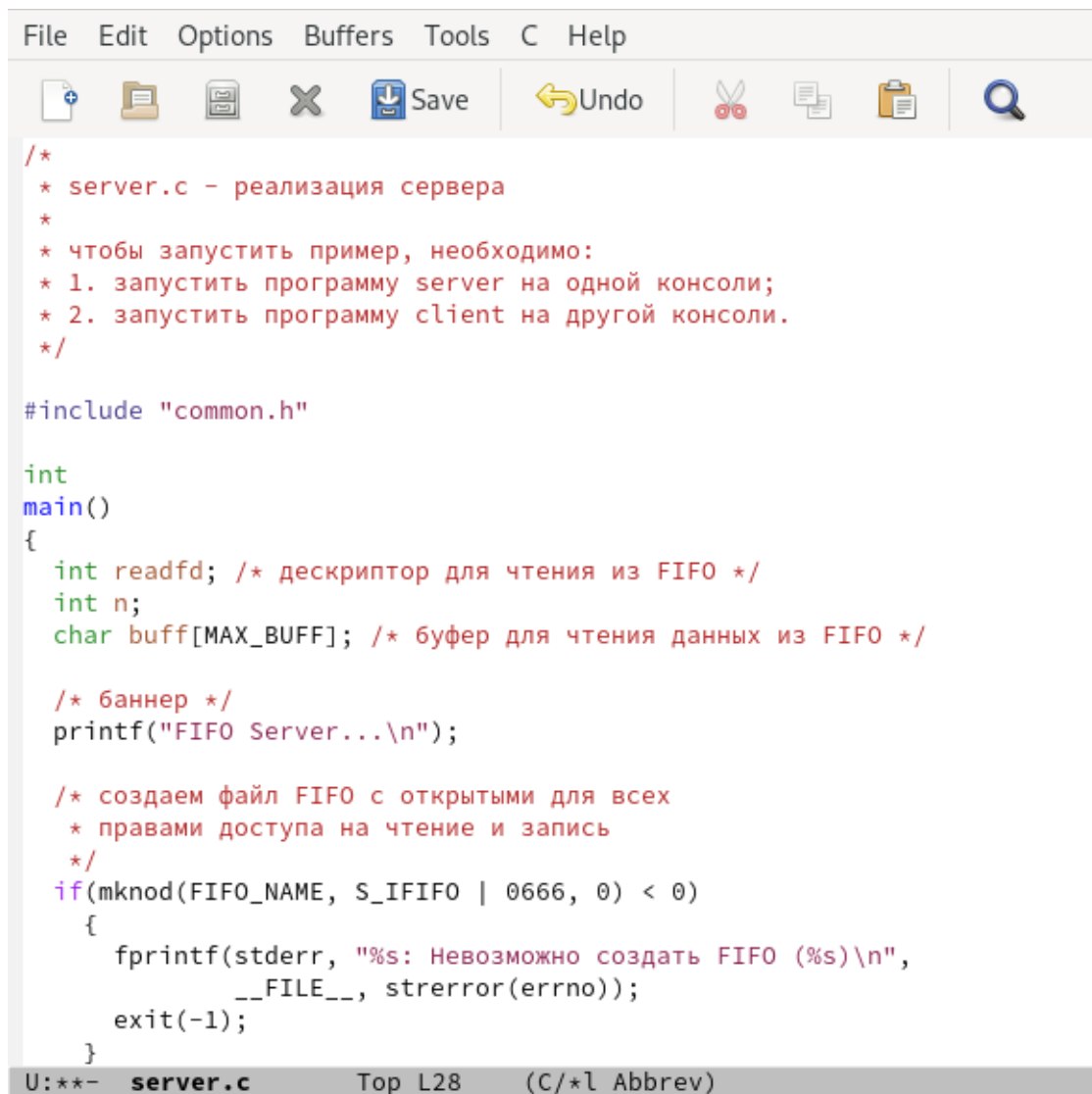
#define FIFO_NAME "/tmp/fifo"
#define MAX_BUFF 80

#endif /* __COMMON_H__ */
```

U:**- common.h All L22 (C/*l Abbrev)

Рис.2

3. Изменим код программы server.c. Добавим цикл while для контроля за временем работы сервера, причем время от начала работы сервера и до настоящего не должно превышать 30 секунд (Рис.3-5).



```
File Edit Options Buffers Tools C Help
[Icons: New, Open, Save, Undo, Redo, Find, etc.]

/*
 * server.c - реализация сервера
 *
 * чтобы запустить пример, необходимо:
 * 1. запустить программу server на одной консоли;
 * 2. запустить программу client на другой консоли.
 */

#include "common.h"

int
main()
{
    int readfd; /* дескриптор для чтения из FIFO */
    int n;
    char buff[MAX_BUFF]; /* буфер для чтения данных из FIFO */

    /* баннер */
    printf("FIFO Server...\n");

    /* создаем файл FIFO с открытыми для всех
     * правами доступа на чтение и запись
     */
    if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)
    {
        fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n",
                __FILE__, strerror(errno));
        exit(-1);
    }

    U:***- server.c Top L28 (C/*l Abbrev)
```

Рис.3

```

    fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n",
        __FILE__, strerror(errno));
    exit(-1);
}

/* откроем FIFO на чтение */
if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
{
    fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
        __FILE__, strerror(errno));
    exit(-2);
}
clock_t start = time(NULL);
/* читаем данные из FIFO и выводим на экран */
while(time(NULL)-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));
            exit(-3);
        }
    }
}

```

U:--- **server.c** 36% L31 (C/*l Abbrev)

Рис.4

```

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

close(readfd); /* закроем FIFO */

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

exit(0);
}

```

U:**~ **server.c** Bot L56 (C/*l Abbrev)

Рис.5

4. Изменим код программы client.c. Добавим цикл for, который отвечает за количество сообщений о текущем времени (4 сообщения), и команду sleep(5) для остановки работы клиента через 5 секунд (Рис.6, 7).

```

/*
 * client.c - реализация клиента
 *
 * чтобы запустить пример, необходимо:
 * 1. запустить программу server на одной консоли;
 * 2. запустить программу client на другой консоли.
 */

#include "common.h"

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

int
main()
{
    int writefd; /* дескриптор для записи в FIFO */
    int msglen;

    /* баннер */
    printf("FIFO Client...\n");

    for (int i=0, i<4, i++)
    {
        if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
        {
            fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
                    __FILE__, strerror(errno));
            exit(-1);
        }
    }
}

```

U:--- **client.c** Top L28 (C/*l Abbrev)

Рис.6

```

    {
        fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
                __FILE__, strerror(errno));
        exit(-1);
    }

    long int Time = time(NULL);
    char* text = ctime(&Time);

    /* передадим сообщение серверу */
    msglen = strlen(MESSAGE);
    if(write(writefd, MESSAGE, msglen) != msglen)
    {
        fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
                __FILE__, strerror(errno));
        exit(-2);
    }
    sleep (5);
}
/* закроем доступ к FIFO */
close(writefd);

exit(0);
}

```

U:--- **client.c** Bot L36 (C/*l Abbrev)

Рис.7

5. Файл Makefile оставляем без изменений (Рис.8).

```

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

```

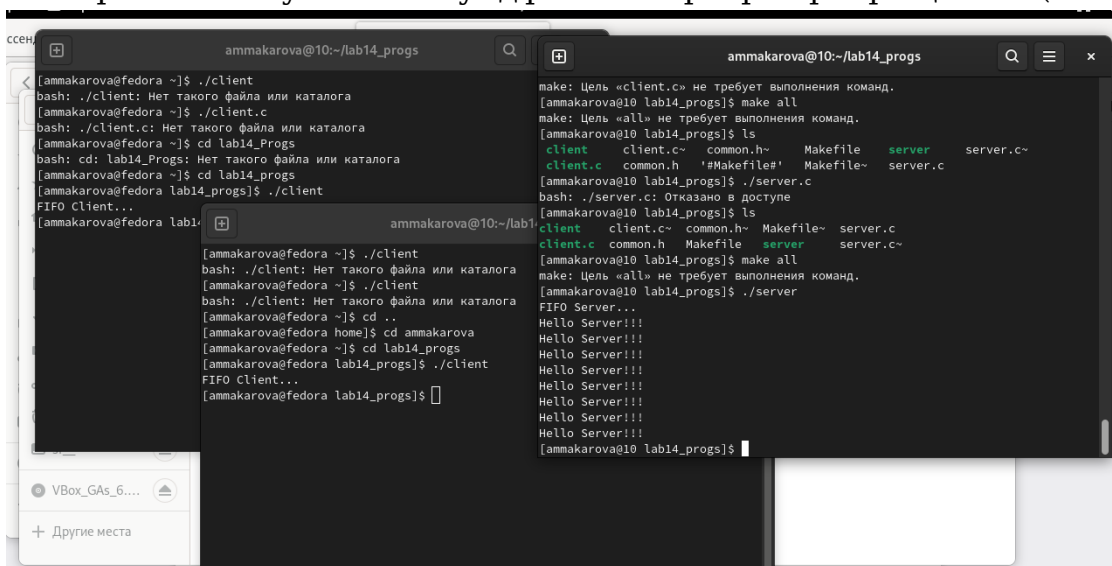
Рис.8

6. С помощью команды `make all` компилируем необходимые файлы для работы программы (Рис.9).

```
[ammakarova@10 lab14_progs]$ make all
gcc client.c -o client
[ammakarova@10 lab14_progs]$ ls
client      client.c~  common.h~  Makefile   server     server.c~
client.c    common.h   '#Makefile#' Makefile~  server.c
```

Рис.9

7. Проверяем работу программы. Открываем три терминала: в первом окне запускаем команду `./server`, а во втором и третьем - `./client`. В результате каждый терминал выводит по 4 сообщения о текущем времени. Спустя 30 секунд работа сервера прекращается (Рис.10).



```
ammakarova@fedora ~]$ ./client
bash: ./client: Нет такого файла или каталога
ammakarova@fedora ~]$ ./client.c
bash: ./client.c: Нет такого файла или каталога
ammakarova@fedora ~]$ cd lab14_Progs
ammakarova@fedora ~]$ cd lab14_progs
ammakarova@fedora lab14_progs]$ ./client
FIFO Client...
ammakarova@fedora lab14_progs]$

ammakarova@10:~/lab14_progs$ ./server
make: Цель «client.c» не требует выполнения команд.
[ammakarova@10 lab14_progs]$ make all
make: Цель «all» не требует выполнения команд.
[ammakarova@10 lab14_progs]$ ls
client      client.c~  common.h~  Makefile   server     server.c~
client.c    common.h   '#Makefile#' Makefile~  server.c
[ammakarova@10 lab14_progs]$ ./server.c
bash: ./server.c: Отказано в доступе
[ammakarova@10 lab14_progs]$ ls
client      client.c~  common.h~  Makefile~  server.c
[ammakarova@10 lab14_progs]$ make all
make: Цель «all» не требует выполнения команд.
[ammakarova@10 lab14_progs]$ ./server
FIFO Server...
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
[ammakarova@10 lab14_progs]$
```

Рис.10

Вывод

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

Контрольные вопросы

1. В чем ключевое отличие именованных каналов от неименованных? Именованные каналы, в отличие от неименованных, могут использоваться неродственными процессами. Они дают вам, по сути, те же возможности, что и неименованные каналы, но с некоторыми преимуществами, присущими обычным файлам. Именованные каналы используют специальную запись в директории для управления правами доступа.

2. Возможно ли создание неименованного канала из командной строки? Вы можете создавать именованные каналы из командной строки и внутри программы. С давних времен программой создания их в командной строке была команда `mknod`:
 - `$ mknod имя_файла p` Однако команды `mknod` нет в списке команд X/Open, поэтому она включена не во все UNIX-подобные системы. Предпочтительнее применять в командной строке
 - `$ mkfifo имя_файла`
3. Возможно ли создание именованного канала из командной строки? Чтобы создать именованный канал из командной строки нужно использовать либо команду «`mknod`», либо команду «`mkfifo`».
4. Опишите функцию языка C, создающую неименованный канал. один из методов межпроцессного взаимодействия (IPC) в операционной системе, который доступен связанным процессам — родительскому и дочернему. Представляется в виде области памяти на внешнем запоминающем устройстве, управляемой операционной системой, которая осуществляет выделение взаимодействующим процессам частей из этой области памяти для совместной работы. Организация данных в канале использует стратегию FIFO, то есть информация, которая первой записана в канал, будет первой прочитана из канала.

Важное отличие неименованного канала от файла заключается в том, что прочитанная информация немедленно удаляется из него и не может быть прочитана повторно. Выполнение вышеперечисленных системных вызовов может переводить процесс в состояние ожидания. Это происходит, если процесс пытается читать данные из пустого канала или писать данные в переполненный канал. Процесс выходит из ожидания, когда в канале появляются данные или когда в канале появляется свободное место, соответственно.

Двустороннее взаимодействие между процессами обычно требует наличия двух неименованных каналов.

5. Опишите функцию языка C, создающую именованный канал. один из методов межпроцессного взаимодействия, расширение понятия конвейера в Unix и подобных ОС. Именованный канал позволяет различным процессам обмениваться данными, даже если программы, выполняющиеся в этих процессах, изначально не были написаны для взаимодействия с другими программами. Это понятие также существует и в Microsoft Windows, хотя там его семантика существенно отличается. Традиционный канал — «безымянный», потому что существует анонимно и только во время выполнения процесса. Именованный канал — существует в системе и после завершения процесса. Он должен быть «отсоединён» или удалён, когда уже не используется. Процессы обычно

подсоединяются к каналу для осуществления взаимодействия между ними.

6. Что будет в случае прочтения из `fifo` меньшего числа байтов, чем находится в канале? Большого числа байтов?
7. Аналогично, что будет в случае записи в `fifo` меньшего числа байтов, чем позволяет буфер? Большого числа байтов?
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.
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.
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.