

Отчет по 14ой лабораторной работе

Лабораторная работа 14

Куркина ЕВгения вячеславовна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводs	17
	Список литературы	18

Список иллюстраций

3.1	Создание файлов	7
3.2	Текст кода фала common.h	8
3.3	Текст кода фала server.c ч1	9
3.4	Текст кода фала server.c ч2	9
3.5	Текст кода фала client.c ч1	10
3.6	Текст кода фала client.c ч2	11
3.7	Текст кода фала Makefile	12
3.8	Результат компиляции	12
3.9	Корреткная работа кода	12
3.10	Проверка команды ./server	13

Список таблиц

1 Цель работы

Здесь приводится формулировка цели лабораторной работы. Формулировки цели для каждой лабораторной работы приведены в методических указаниях.

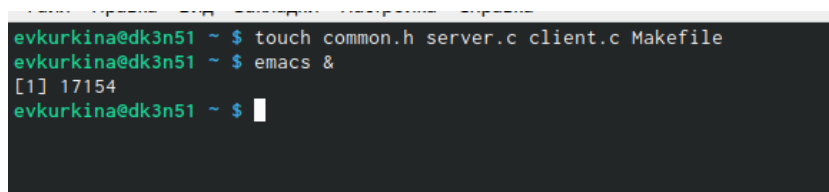
Цель данной лабораторной работы — Приобрести прктические навыки работы с именованными каналами.

2 Задание

Здесь приводится описание задания в соответствии с рекомендациями методического пособия и выданным вариантом.

3 Выполнение лабораторной работы

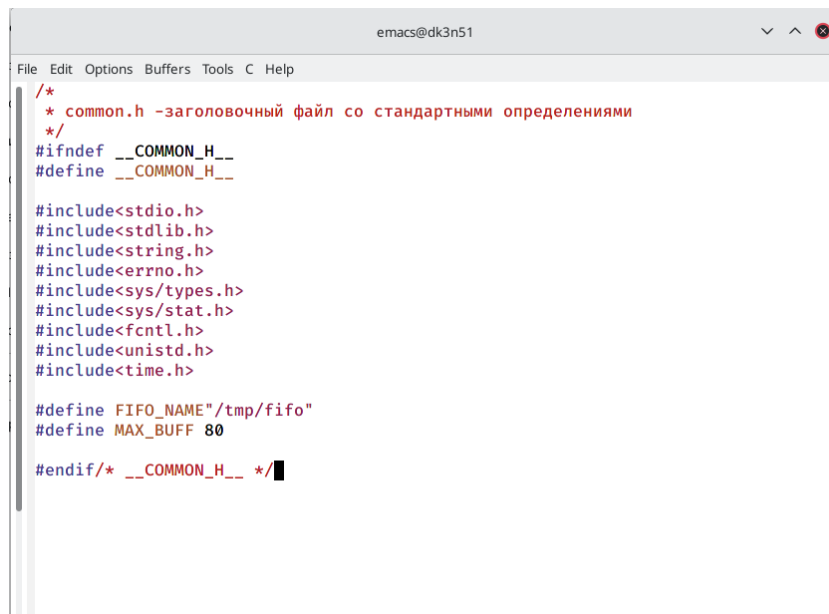
1)Создала необходимые файлы для дальнейшей работы командой touch common.h server.c client.c Makefile, а затем перешла в emacs. (рис. 3.1).



```
evkurkina@dk3n51 ~ $ touch common.h server.c client.c Makefile
evkurkina@dk3n51 ~ $ emacs &
[1] 17154
evkurkina@dk3n51 ~ $
```

Рис. 3.1: Создание файлов

2)Я изменила коды программ , которые были представлены в указание к работе. В файл common.h добавила стандартные заголовочные файлы unistd.h и time.h,которые необходимы для работы кодов других файлов. Common.h предназначен для заголовочных файлов (рис. 3.2) .



```
/*
 * 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__ */
```

Рис. 3.2: Текст кода файла common.h

Далее в файле server.c объявила цикл while для контроля работы времени сервера. Разница во времени текущем и временем начала работы не должна быть более 30 секунд (рис. 3.3)(рис. 3.4) .


```
Приложения  Места  GNU Emacs
emacs
File Edit Options Buffers Tools C Help

/*
 *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 */

    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 | O_NONBLOCK)) < 0)
    {
        printf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
                __FILE__, strerror(errno));
        exit(-2);
    }

    clock_t start = time(NULL);
    while (time(NULL) - start < 30)
    {
        while ((n = read(readfd, buff, MAX_BUFF)) > 0)
        {
```

Рис. 3.3: Текст кода файла server.c ч1

```
File Edit Options Buffers Tools C Help
while ((n = read(readfd, buff, MAX_BUFF)) > 0)
{
    if(write(1, buff, n) != n)
    {
        printf(stderr, "%s: Ошибка вывода (%s)\n",
                __FILE__, strerror(errno));
        exit(-3);
    }
}

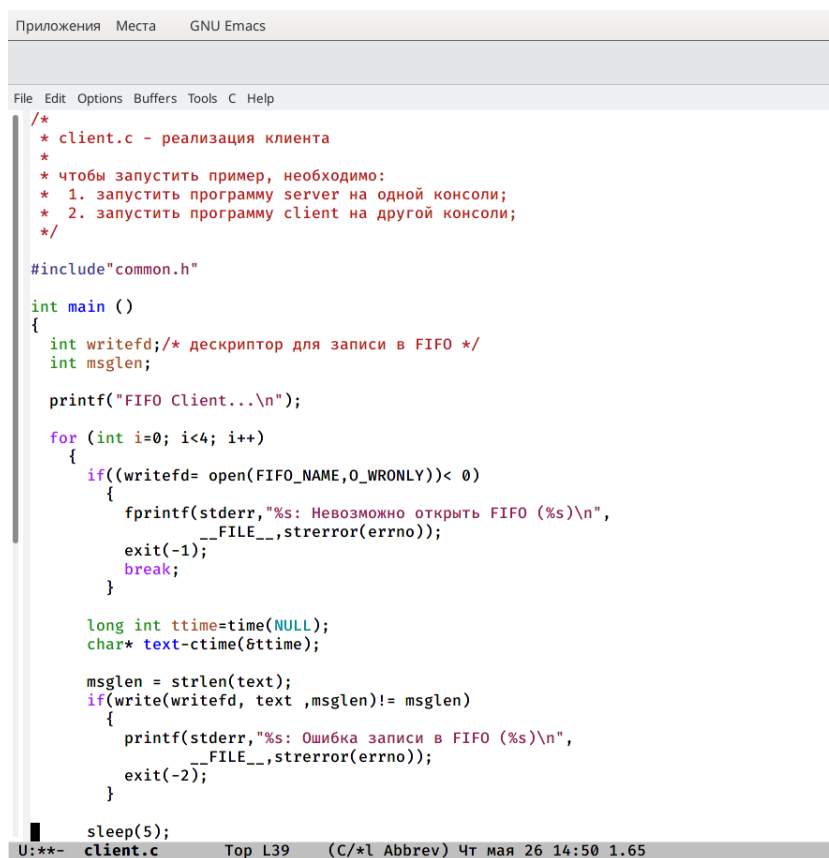
close(readfd); /* закроем FIFO */

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

Рис. 3.4: Текст кода файла server.c ч2

В файл client.c добавила цикл, отвечающий за количество сообщений о теку-

щем времени, которое получается в результате выполнения команд (рис. 3.5)(рис. 3.6).



```
/*
 * 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"

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

    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);
            break;
        }

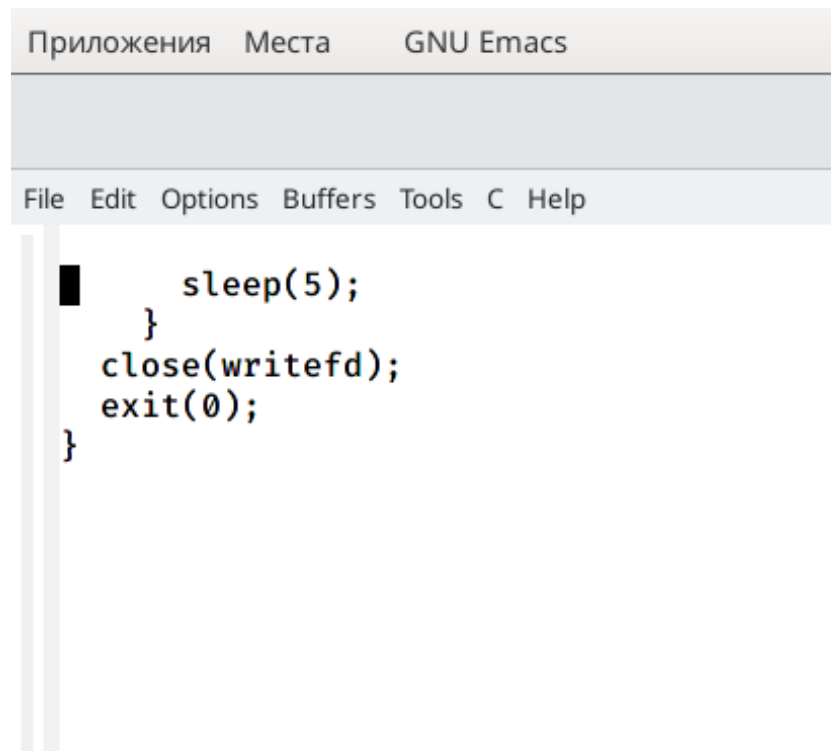
        long int ttime=time(NULL);
        char* text=ctime(&ttime);

        msglen = strlen(text);
        if(write(writefd, text ,msglen)!= msglen)
        {
            printf(stderr,"%s: Ошибка записи в FIFO (%s)\n",
                __FILE__,strerror(errno));
            exit(-2);
        }

        sleep(5);
    }
}
```

U:*** client.c Top L39 (C/*l Abbrev) Чт мая 26 14:50 1.65

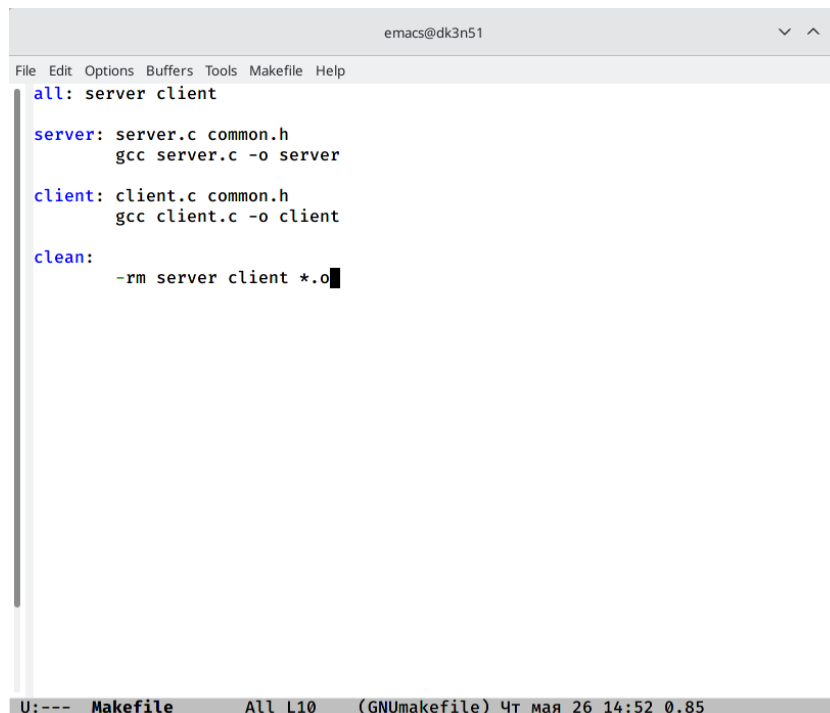
Рис. 3.5: Текст кода файла client.c ч1

A screenshot of the GNU Emacs editor interface. The title bar at the top contains the text "Приложения Места GNU Emacs". Below it is a menu bar with the items "File", "Edit", "Options", "Buffers", "Tools", "C", and "Help". The main editing area displays C code with a black cursor at the start of the first line. The code is as follows:

```
    sleep(5);  
    }  
    close(writefd);  
    exit(0);  
}
```

Рис. 3.6: Текст кода файла client.c ч2

Makefile оставила без изменений (рис. 3.7).



```
emacs@dk3n51
File Edit Options Buffers Tools Makefile Help

all: server client

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

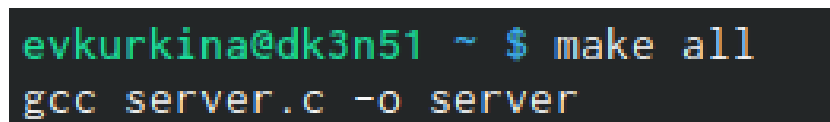
client: client.c common.h
    gcc client.c -o client

clean:
    -rm server client *.o

U:--- Makefile All L10 (GNUmakefile) Чт мая 26 14:52 0.85
```

Рис. 3.7: Текст кода фала Makefile

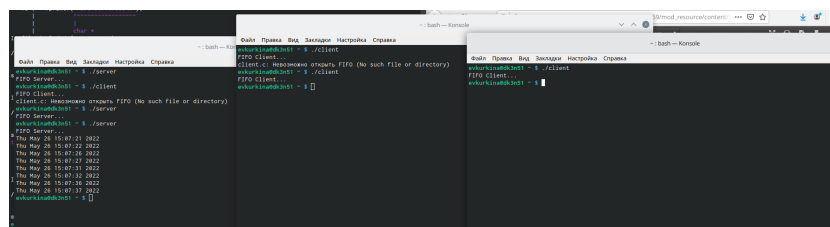
3) После того, как написала все коды, с помощью команды `make all`, скомпилировала необходимые файлы (рис. 3.8) .



```
evkurkina@dk3n51 ~ $ make all
gcc server.c -o server
```

Рис. 3.8: Результат компиляции

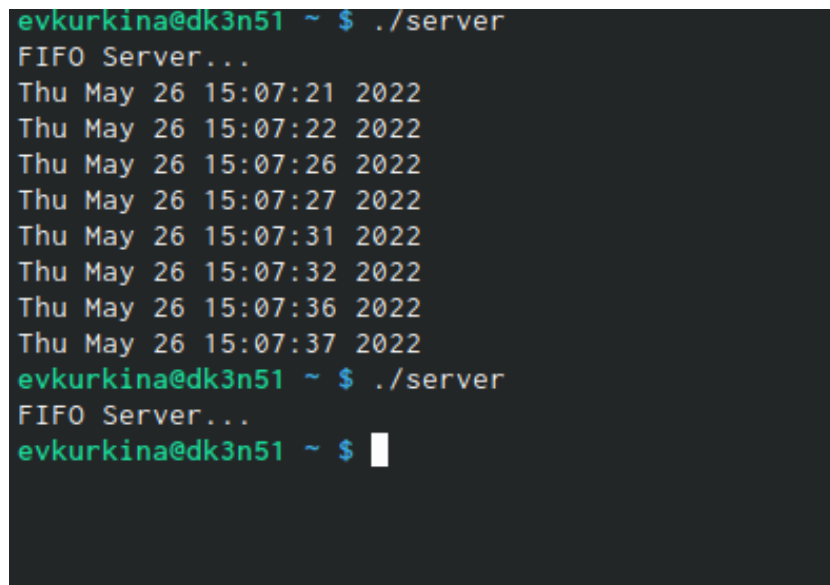
4) Далее я открыла 3 консоли и проверила работу написанного кода. В первом окне запустила сервер, а в двух других клиент, убедилась в корректной работе кода (рис. 3.9).



```
evkurkina@dk3n51 ~ $ ./server
FIFO Server:
Thu May 26 15:07:11 2022
Thu May 26 15:07:22 2022
Thu May 26 15:07:26 2022
Thu May 26 15:07:27 2022
Thu May 26 15:07:28 2022
Thu May 26 15:07:28 2022
Thu May 26 15:07:28 2022
evkurkina@dk3n51 ~ $ ./client
FIFO Client:
evkurkina@dk3n51 ~ $ ./client
FIFO Client:
evkurkina@dk3n51 ~ $
```

Рис. 3.9: Корреткная работа кода

- 5) После, убедилась в том, что код работает не более 30 секунд, отдельно запустила команду `./server` (рис. 3.10).



```
evkurkina@dk3n51 ~ $ ./server
FIFO Server...
Thu May 26 15:07:21 2022
Thu May 26 15:07:22 2022
Thu May 26 15:07:26 2022
Thu May 26 15:07:27 2022
Thu May 26 15:07:31 2022
Thu May 26 15:07:32 2022
Thu May 26 15:07:36 2022
Thu May 26 15:07:37 2022
evkurkina@dk3n51 ~ $ ./server
FIFO Server...
evkurkina@dk3n51 ~ $ █
```

Рис. 3.10: Проверка команды `./server`

- 6) 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). Чтобы создать неименованный канал из командной строки нужно использовать символ `|`, служащий для объединения двух и более процессов: `процесс_1 | процесс_2 | процесс_3...`

3). Чтобы создать именованный канал из командной строки нужно использовать либо команду `«mknod»`, либо команду `«mkfifo»`.

4). Неименованный канал является средством взаимодействия между связанными процессами – родительским и дочерним. Родительский процесс создает канал при помощи системного вызова: `«int pipe(int fd[2]);»`. Массив из двух целых

5). Файлы именованных каналов создаются функцией `mkfifo()` или функцией `mknod`:

1. После создания файла канала процессы, участвующие в обмене данными, должны отк

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=ERRPIPE`) (если процесс не установил обработки сигнала `SIGPIPE`, производится обработка по умолчанию – процесс завершается).

8). Количество процессов, которые могут параллельно присоединяться к любому концу канала, не ограничено. 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. Затем, когда выполнение процесса А возобновляется, он записывает оставшиеся $X-Y$ байтов данных в канал. В результате данные в канал записываются поочередно двумя процессами. Аналогичным образом, если два (или более) процесса одновременно попытаются прочитать данные из канала, может случиться так, что каждый из них прочитает только часть необходимых данных.

9). Функция `write` записывает байты `count` из буфера `buffer` в файл, связанный с `handle`. Операции `write` начинаются с текущей позиции указателя на файл (указатель ассоциирован с заданным файлом). Если файл открыт для добавления, операции выполняются в конец файла. После осуществления операций записи указатель на файл (если он есть) увеличивается на количество действительно записанных байтов. Функция `write` возвращает число действительно записанных байтов. Возвращаемое значение должно быть положительным, но меньше числа `count` (например, когда размер для записи `count` байтов выходит за пре-

делу пространства на диске). Возвращаемое значение -1 указывает на ошибку; errno устанавливается в одно из следующих значений: EACCES – файл открыт для чтения или закрыт для записи, EBADF – неверный handle-р файла, ENOSPC – на устройстве нет свободного места. Единица в вызове функции write в программе server.c означает идентификатор (дескриптор потока) стандартного потока вывода.

10). Прототип функции strerror: «char * strerror(int errornum);». Функция strerror интерпретирует номер ошибки, передаваемый в функцию в качестве аргумента – errornum, в понятное для человека текстовое сообщение (строку). Откуда берутся эти ошибки? Ошибки эти возникают при вызове функций стандартных Си-библиотек. То есть хорошим тоном программирования будет – использование этой функции в паре с другой, и если возникнет ошибка, то пользователь или программист поймет, как исправить ошибку, прочитав сообщение функции strerror. 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 Выводs

Во время выполнения данной лабораторной работы я приобрела практические навыки работы с именованными каналами.

Список литературы