

Отчёт к лабораторной работе №15

Именованные каналы.

Кекишева Анастасия Дмитриевна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Вывод	14
5	Библиография	15
6	Контрольные вопросы	16

Список таблиц

Список иллюстраций

3.1	Файл common.h	7
3.2	Файл server.c: время работы 30 сек	8
3.3	Файл client.c	9
3.4	Файл client02.c	10
3.5	Makefile	11
3.6	Компиляция файлов	11
3.7	Результат работы программы 1	11
3.8	Файл server.c: время работы 20 сек	12
3.9	Результат работы программы 2	13

1 Цель работы

Приобретение практических навыков работы с именованными каналами.

2 Задание

Изучить приведённые в тексте программы `server.c` и `client.c`. Взяв данные примеры за образец, напишите аналогичные программы, внося следующие изменения:

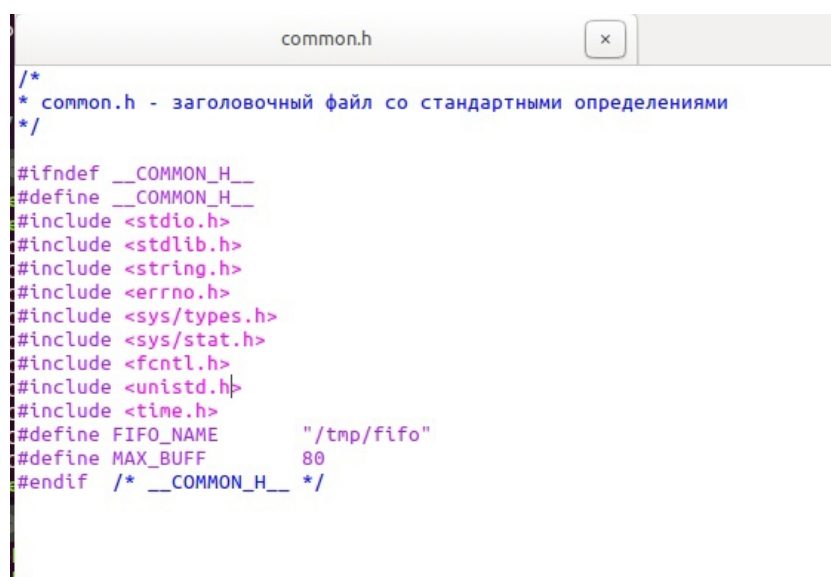
- Работает не 1 клиент, а несколько (например, два).
- Клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд). Используйте функцию `sleep()` для приостановки работы клиента.
- Сервер работает не бесконечно, а прекращает работу через некоторое время (например, 30 сек). Используйте функцию `clock()` для определения времени работы сервера. Что будет в случае, если сервер завершит работу, не закрыв канал?

3 Выполнение лабораторной работы

Перед выполнением лабораторной работы я хорошо ознакомилась с теоретическим материалом для её выполнения Ссылка 1

Во-первых, я изучила приведённые в тексте программы server.c и client.c. Взяв данные примеры за образец, написала аналогичные программы, внося следующие изменения:

- Работает не 1 клиент, а несколько (например, два).
- Клиенты передают текущее время с некоторой периодичностью, я взяла что первый работает 3 сек, а второй 5, для приостановки работы клиента я использовала функцию sleep().

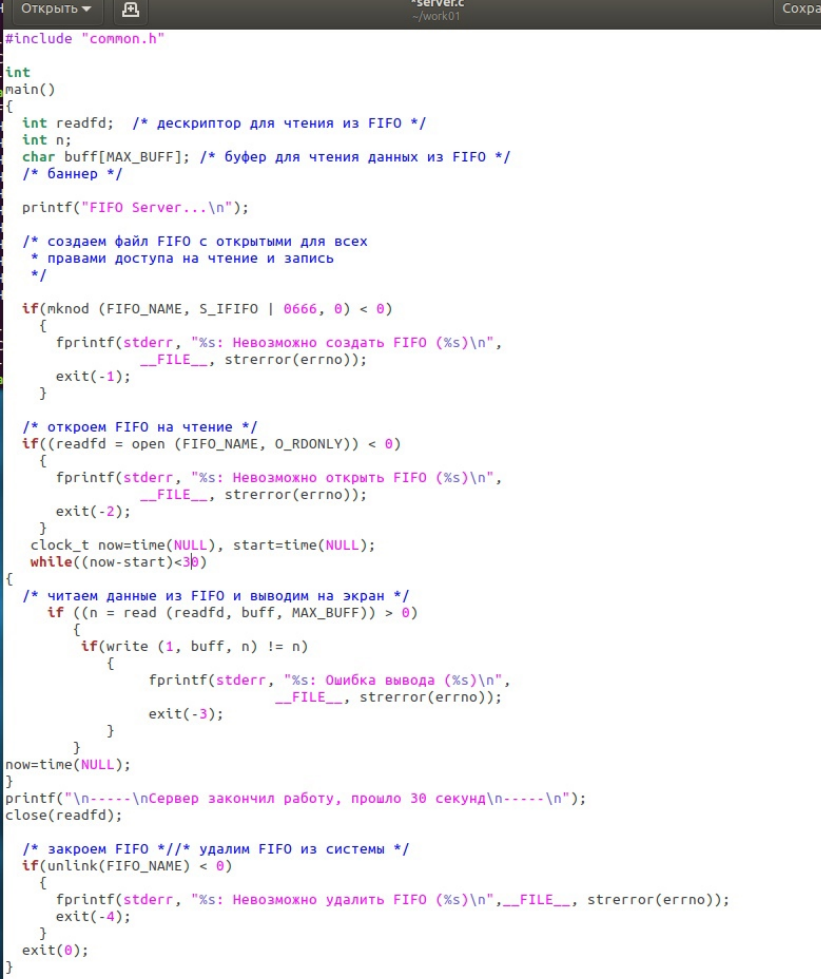
A screenshot of a code editor window titled 'common.h'. The code is a C header file. It starts with a multi-line comment: '/* * common.h - заголовочный файл со стандартными определениями */'. This is followed by a preprocessor guard: '#ifndef __COMMON_H__', then '#define __COMMON_H__'. Next are several include statements: '#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>', and '#include <time.h>'. Then there are two macro definitions: '#define FIFO_NAME "/tmp/fifo"' and '#define MAX_BUFF 80'. The file ends with '#endif /* __COMMON_H__ */'.

```
/*
 * 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. 3.1: Файл common.h

В файл `common.h` я добавила две библиотеки `time.h` и `unistd.h` (рис. 3.1). Первую для времени, и вторую для функций `read`, `wright`, `close` и т.д.



```
#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
     */

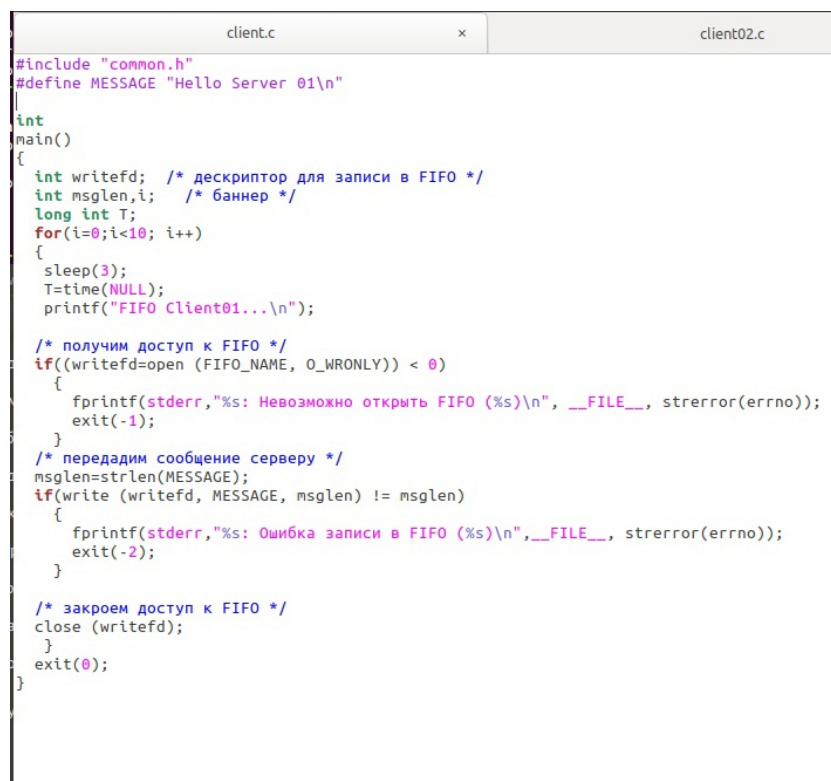
    if(mknod (FIFO_NAME, S_IFIFO | 0666, 0) < 0)
    {
        fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n",
            __FILE__, strerror(errno));
        exit(-1);
    }

    /* 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 now=time(NULL), start=time(NULL);
    while((now-start)<30)
    {
        /* читаем данные из FIFO и выводим на экран */
        if ((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);
            }
        }
        now=time(NULL);
    }
    printf("\n-----\nСервер закончил работу, прошло 30 секунд\n-----\n");
    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);
}
```

Рис. 3.2: Файл `server.c`: время работы 30 сек

В файле `server.c` добавила функцию `clock`, присваиваю переменным `start` и `now` значение `time(NULL)`. Также, создала цикл `while` установила время работы 50 сек., в конце цикла мы присваиваем значение `time(NULL)` переменной `now`. вывожу на экран сообщение о том, что сервер закончил свою работу и время его работы.



```
client.c
x
client02.c

#include "common.h"
#define MESSAGE "Hello Server 01\n"

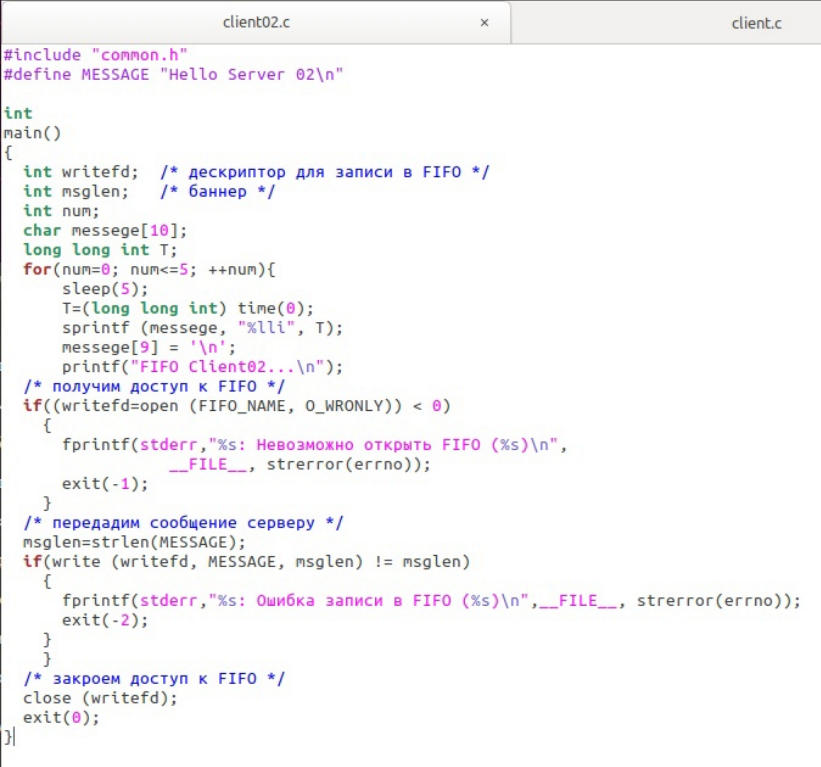
int
main()
{
    int writefd; /* дескриптор для записи в FIFO */
    int msglen,i; /* баннер */
    long int T;
    for(i=0;i<10; i++)
    {
        sleep(3);
        T=time(NULL);
        printf("FIFO client01...\n");

        /* получим доступ к FIFO */
        if((writefd=open (FIFO_NAME, O_WRONLY)) < 0)
        {
            fprintf(stderr,"%s: Невозможно открыть FIFO (%s)\n", __FILE__, strerror(errno));
            exit(-1);
        }
        /* 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);
        }

        /* закроем доступ к FIFO */
        close (writefd);
    }
    exit(0);
}
```

Рис. 3.3: Файл client.c

В файле client.c я ввела дополнительную переменную T для времени, а также сделала цикл for, в котором я определяю сколько раз в первый клиент будет обработан сервером, а также функцию sleep(3), где задаю время ожидания 3 сек, и вывод на экран, что это клиент первый. Также передаю сообщение серверу, что это первый клиент (#define MESSAGE "Hello Server 01").



```

client02.c
x
client.c

#include "common.h"
#define MESSAGE "Hello Server 02\n"

int
main()
{
    int writefd; /* дескриптор для записи в FIFO */
    int msglen; /* баннер */
    int num;
    char messege[10];
    long long int T;
    for(num=0; num<=5; ++num){
        sleep(5);
        T=(long long int) time(0);
        sprintf (messege, "%lli", T);
        messege[9] = '\n';
        printf("FIFO Client02...\n");
        /* получим доступ к FIFO */
        if((writefd=open (FIFO_NAME, O_WRONLY)) < 0)
        {
            fprintf(stderr,"%s: Невозможно открыть FIFO (%s)\n",
                __FILE__, strerror(errno));
            exit(-1);
        }
        /* 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);
        }
        /* закроем доступ к FIFO */
        close (writefd);
        exit(0);
    }
}

```

Рис. 3.4: Файл client02.c

В файле client02.c () я также ввела дополнительную переменную T для времени, массив messege[10], также сделала цикл for, в котором я определяю сколько раз в клиент будет обработан сервером, а также функцию sleep(5), где задаю время ожидания 5 сек, и вывод на экран, что это второй клиент. Функцией sprintf() произвожу вывод в массив. Также передаю сообщение серверу, что это второй клиент (#define MESSAGE "Hello Server 02").

Далее запустила в одной консоли server, во второй client, в третьей client02. Сервер обработал клиентов необходимое количество раз и по прошествии 30 сек закончил работу (рис 3.7).



```

client02.c x server.c x common.h x cl
#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);
    }

    /* 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 now=time(NULL), start=time(NULL);
    while((now-start)<20)
    {
        /* читаем данные из FIFO и выводим на экран */
        if ((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);
            }
        }
    }
    now=time(NULL);
}
printf("\n-----\nСервер закончил работу, прошло 20 секунд\n-----\n");
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);
}

```

Рис. 3.8: Файл server.c: время работы 20 сек

[illegible]

Рис. 3.9: Результат работы программы 2

Изменила время работы сервера на 20 сек (рис 3.8). В это раз сервер не успел обработать всех. Если сервер завершит работу, не закрыв канал, для клиентов выйдет сообщение о том, что невозможно открыть FIFO (рис 3.9).

4 Вывод

Я приобрела практические навыки работы с именованными каналами, написала программы, с требующимися изменениями.

5 Библиография

1. Ссылка 1

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. Создание неименованного канала из командной строки невозможно.
3. Создание именованного канала из командной строки возможно.
4. `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. `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`).

6. 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.

7. 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, гарантировано атомарно. Это означает, что в случае, когда несколько процессов одновременно записывают в канал, порции данных от этих процессов не перемешиваются.
8. 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.
9. `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.
10. Строковая функция `strerror` - функция языков C/C++, транслирующая код ошибки, который обычно хранится в глобальной переменной `errno`, в сообщении 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.