

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Лабораторная работа No 14. Именованные каналы

Абдуллина Ляйсан Раисовна НПИбд-01-21

Содержание

1	Цель работы	4
2	Теоретическое введение	5
3	Выполнение лабораторной работы	6
3.1	1	6
4	Контрольные вопросы	10
5	Выводы	14
6	Список литературы	15

Список иллюстраций

3.1	Изменный код client.c	7
3.2	Изменный код common.h	7
3.3	Изменный код server.c	8
3.4	Код Makefile	8
3.5	Компийция	9
3.6	Проверка программ - успешно	9

1 Цель работы

Приобретение практических навыков работы с именованными каналами.

2 Теоретическое введение

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.

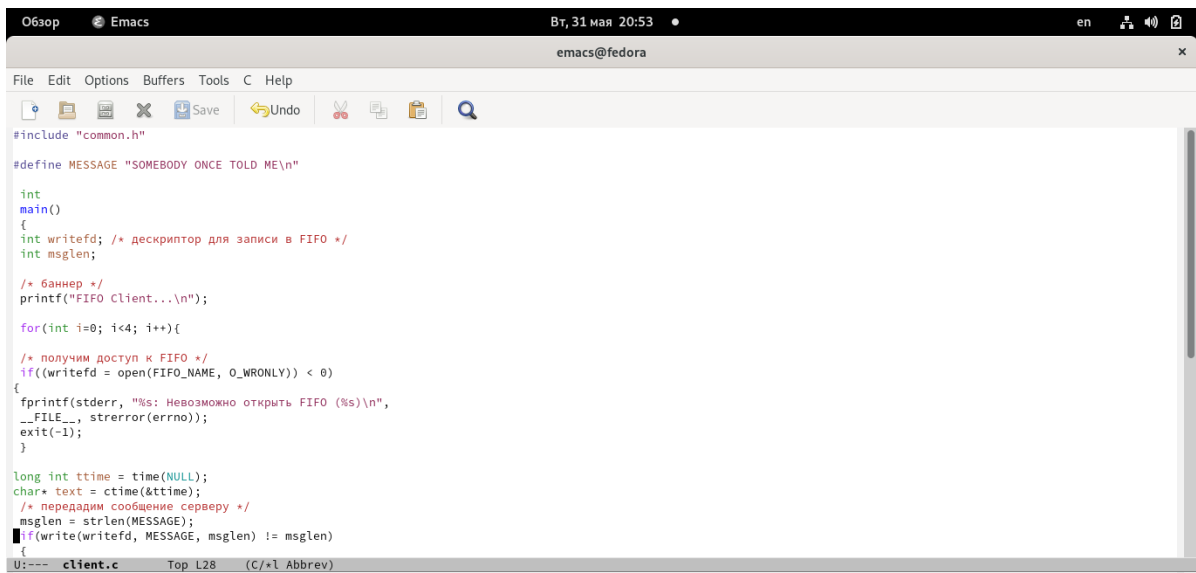
3 Выполнение лабораторной работы

3.1 1

Создадим подкаталог prog1 через команду `mkdir` и нужные нам файлы “common.h”, “server.c”, “client.c” и “Makefile” через команду `touch`

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

1. Работает не 1 клиент, а несколько (например, два).
 2. Клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд). Используем функцию `sleep()` для приостановки работы клиента.
 3. Сервер работает не бесконечно, а прекращает работу через некоторое время (например, 30 сек). Используем функцию `clock()` для определения времени работы сервера.
- В файл `client.c` добавим цикл, который отвечает за количество сообщений о текущем времени (4 сообщения), которое получается в результате выполнения команд, и команду `sleep(5)` для приостановки работы клиента на 5 секунд. (скриншот 3.1)



```
#include "common.h"

#define MESSAGE "SOMEBODY ONCE TOLD ME\n"

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

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

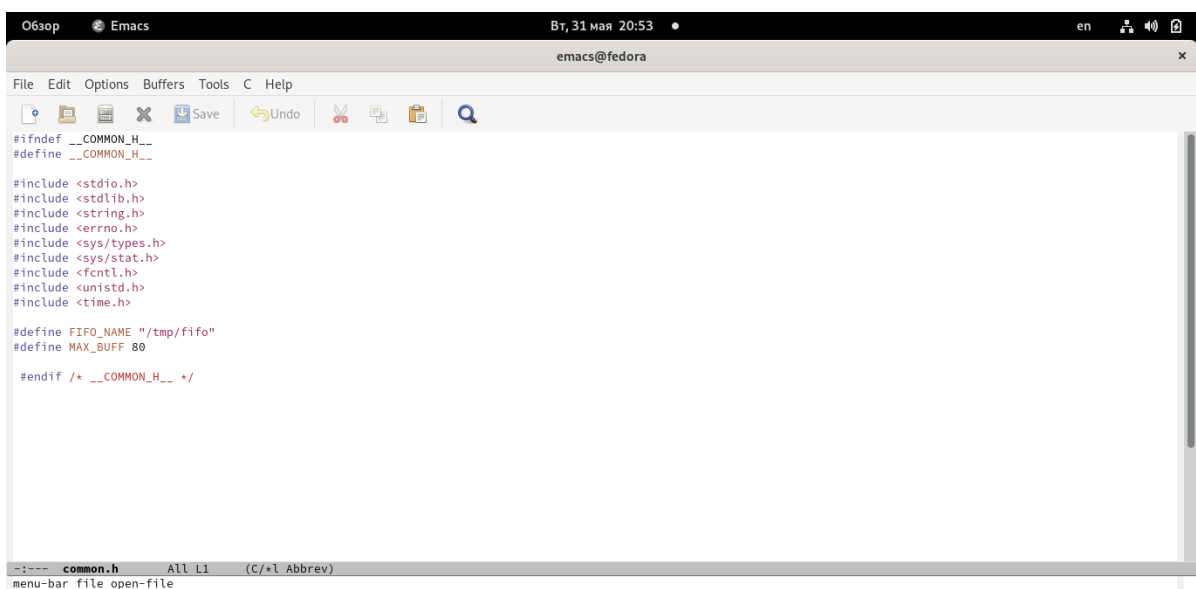
    for(int i=0; i<4; i++){

        /* 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);
        }

        long int ttime = time(NULL);
        char* text = ctime(&ttime);
        /* передадим сообщение серверу */
        msglen = strlen(MESSAGE);
        if(write(writefd, MESSAGE, msglen) != msglen)
        {
            /* ... */
        }
    }
}
```

Скриншот 3.1: Изменный код client.c

- В файл common.h добавим стандартные заголовочные файлы unistd.h и time.h, необходимые для работы кодов других файлов. Common.h предназначен для заголовочных файлов, чтобы в остальных программах их не прописывать каждый раз (скриншот 3.2)



```
#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__ */
```

Скриншот 3.2: Изменный код common.h

- В файл `server.c` добавим цикл `while` для контроля за временем работы сервера. Разница между текущим временем `time(NULL)` и временем начала работы `clock_t start=time(NULL)` (инициализация до цикла) не должна превышать 30 секунд. (скриншот 3.3)

```

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);
while(time(NULL)-start < 30)
{
    /* читаем данные из FIFO и выводим на экран */
    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 */
U:--- server.c 31% L31 (C/*l Abbrev)

```

Скриншот 3.3: Измененный код `server.c`

- `Makefile` (файл для сборки) не изменяем (скриншот 3.4)

```

all: server client

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

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

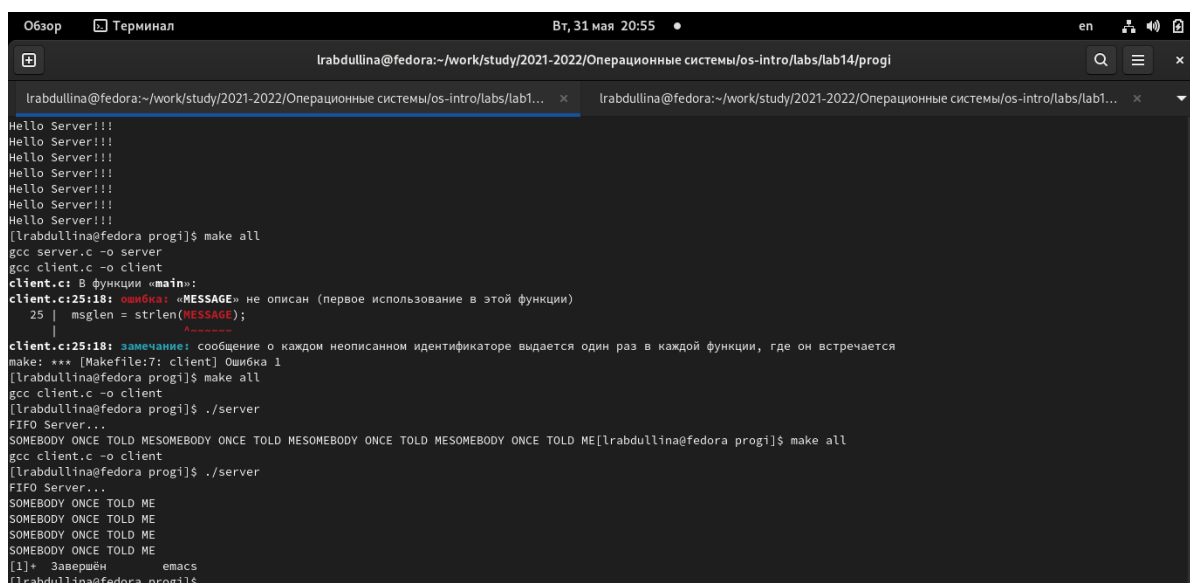
clean:
    -rm server client *.o

-:--- Makefile All L1 (GNUmakefile)
menu-bar file open-file

```

Скриншот 3.4: Код `Makefile`

Далее мы компилировали наши файлы через команду `make all` (скриншот 3.5)

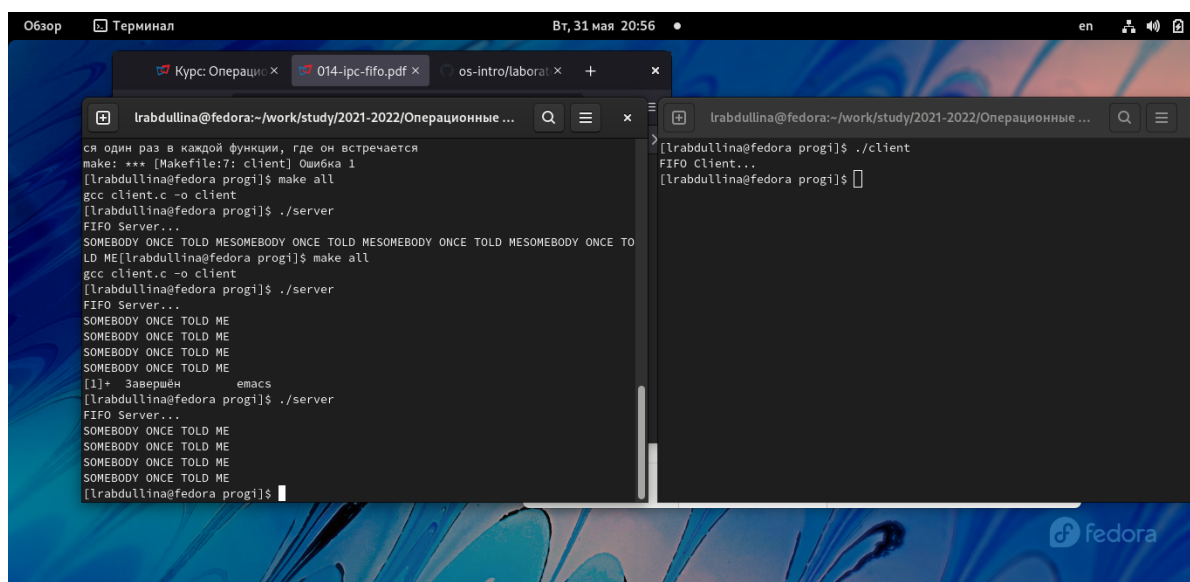


```
lrabduullina@fedora:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab14/progi
lrabduullina@fedora:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab14/progi$ make all
gcc server.c -o server
gcc client.c -o client
client.c: В функции «main»:
client.c:25:18: ошибка: «MESSAGE» не описан (первое использование в этой функции)
   25 |     msglen = strlen(MESSAGE);
      |                      ^
client.c:25:18: замечание: сообщение о каждом неопisanном идентификаторе выдается один раз в каждой функции, где он встречается
make: *** [Makefile:7: client] Ошибка 1
lrabduullina@fedora:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab14/progi$ make all
gcc client.c -o client
lrabduullina@fedora:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab14/progi$ ./server
FIFO Server...
SOMEBODY ONCE TOLD MESOMEBODY ONCE TOLD MESOMEBODY ONCE TOLD MESOMEBODY ONCE TOLD ME[lrabduullina@fedora:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab14/progi]$ make all
gcc client.c -o client
lrabduullina@fedora:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab14/progi$ ./server
FIFO Server...
SOMEBODY ONCE TOLD ME
SOMEBODY ONCE TOLD ME
SOMEBODY ONCE TOLD ME
SOMEBODY ONCE TOLD ME
[1]+  Завершён      emacs
lrabduullina@fedora:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab14/progi$
```

Скриншот 3.5: Компиляция

После чего открываем 2 терминала и проверяем работу наших программ. Вводим в первый терминал `./server`, во второй - `./client`

Программа останавливается после 30 секунд, следовательно работают успешно.(скриншот 3.6)



```
lrabduullina@fedora:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab14/progi$ ./server
FIFO Server...
SOMEBODY ONCE TOLD MESOMEBODY ONCE TOLD MESOMEBODY ONCE TOLD MESOMEBODY ONCE TOLD ME
LD ME[lrabduullina@fedora:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab14/progi]$ make all
gcc client.c -o client
lrabduullina@fedora:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab14/progi$ ./server
FIFO Server...
SOMEBODY ONCE TOLD ME
SOMEBODY ONCE TOLD ME
SOMEBODY ONCE TOLD ME
SOMEBODY ONCE TOLD ME
[1]+  Завершён      emacs
lrabduullina@fedora:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab14/progi$ ./server
FIFO Server...
SOMEBODY ONCE TOLD ME
SOMEBODY ONCE TOLD ME
SOMEBODY ONCE TOLD ME
SOMEBODY ONCE TOLD ME
lrabduullina@fedora:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab14/progi$

lrabduullina@fedora:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab14/progi$ ./client
FIFO Client...
lrabduullina@fedora:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab14/progi$
```

Скриншот 3.6: Проверка программ - успешно

4 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 чисел является выходным параметром этого системного вызова. Если вызов выполнен нормально, то этот массив содержит два файловых дескриптора. `fd[0]` является дескриптором для чтения из канала, `fd1` – дескриптором для записи в канал. Когда процесс порождает другой процесс, дескрипторы родительского процесса наследуются дочерним процессом, и, таким образом, прокладывается трубопровод между двумя процессами. Естественно, что один из процессов использует канал только для чтения, а другой – только для записи. Поэтому, если, например, через канал должны

передаваться данные из родительского процесса в дочерний, родительский процесс сразу после запуска дочернего процесса закрывает дескриптор канала для чтения, а дочерний процесс закрывает дескриптор для записи. Если нужен двунаправленный обмен данными между процессами, то родительский процесс создает два канала, один из которых используется для передачи данных в одну сторону, а другой – в другую.

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

- «`int mkfifo(const char *pathname, mode_t mode);`», где первый параметр – путь, где будет располагаться FIFO (имя файла, идентифицирующего канал), второй параметр определяет режим работы с FIFO (маска прав доступа к файлу),
- «`mknod (namefile, IFIFO | 0666, 0)`», где `namefile` – имя канала, `0666` – к каналу разрешен доступ на запись и на чтение любому запросившему процессу),
- «`int mknod(const char *pathname, mode_t mode, dev_t dev);`».

Функция `mkfifo()` создает канал и файл соответствующего типа. Если указанный файл канала уже существует, `mkfifo()` возвращает -1. После создания файла канала процессы, участвующие в обмене данными, должны открыть этот файл либо для записи, либо для чтения.

6. При чтении меньшего числа байтов, чем находится в канале или FIFO, возвращается требуемое число байтов, остаток сохраняется для последующих чтений. 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. 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 пределы 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.

5 Выводы

В ходе лабораторной работы мы приобрели практические навыки работы с именованными каналами

6 Список литературы

<https://esystem.rudn.ru/course/view.php?id=5790> :::