

Лабораторная работа №15 по предмету
Операционные системы

Группа НПМбв-01-19

Бондаренко Артем Федорович

Содержание

Цель работы	5
Задание	6
Выполнение лабораторной работы	7
Выводы	14
Ответы на контрольные вопросы	15

Список иллюстраций

1	Создание директории	7
2	Создание файлов командой touch	7
3	Код файла common.h	8
4	Код файла client.c	8
5	Код файла server.c	9
6	Работа исполняемого файла Server в терминале	10
7	Работа исполняемого файла Client в терминале	10
8	Изменение файла common.h	11
9	Изменение файла client.c	11
10	Изменение файла server.c	12
11	Демонстрация работы получившегося файла server в терминале . . .	13
12	Демонстрация работы получившегося файла client в терминале	13

Список таблиц

Цель работы

Приобретение практических навыков работы с именованными каналами.

Задание

Изучите приведённые в тексте программы `server.c` и `client.c`. Взяв данные примеры за образец, напишите 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). Используйте функцию `sleep()` для приостановки работы клиента.
3. 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?

Выполнение лабораторной работы

Изучил приведённые в тексте программы `server.c`, `client.c` и `common.h` после чего создал специальную директорию под тестирование этих файлов и назвал её `server`. (Ссылка: Рис.1)

```
[afbond@fedora ~]$ mkdir server2  
[afbond@fedora ~]$ cd server2
```

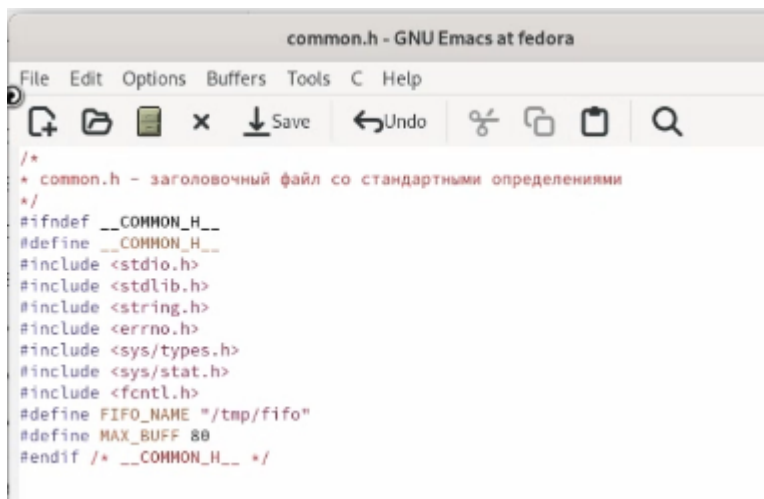
Рис. 1: Создание директории

Создал аналогичные программам в лабораторной работе файлы посредством терминала с помощью команды `touch`. (Ссылка: Рис.2)

```
[afbond@fedora ~]$ cd server2  
[afbond@fedora server2]$ touch common.h  
[afbond@fedora server2]$ touch server.c  
[afbond@fedora server2]$ touch client.c  
[afbond@fedora server2]$ touch Makefile
```

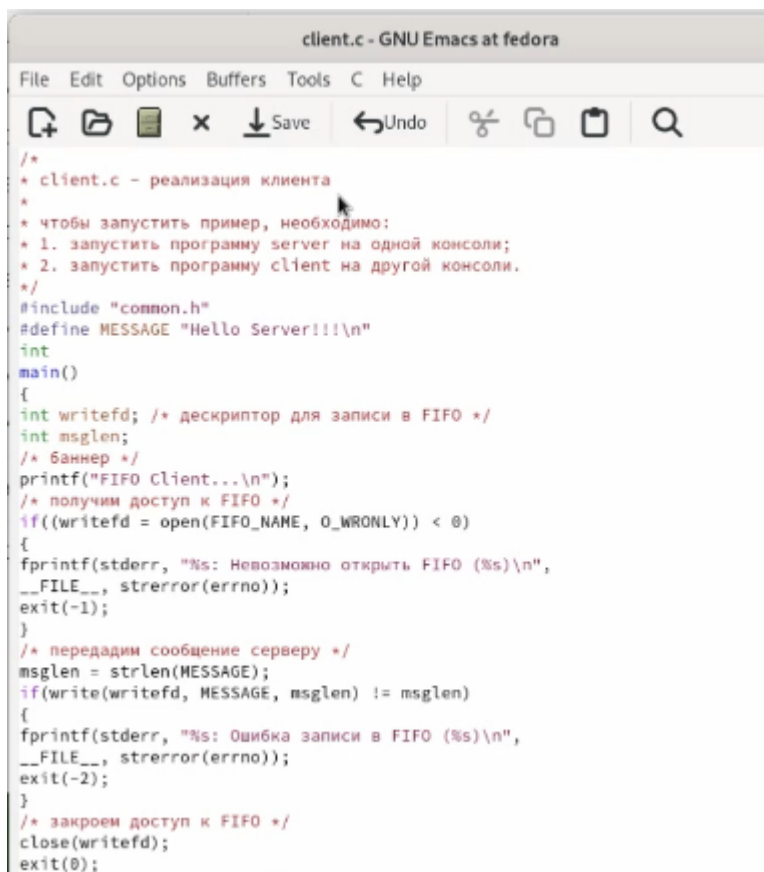
Рис. 2: Создание файлов командой `touch`

После чего в каждый из этих файлов скопировал программный код из лабораторной работы, внося небольшие корректировки. (Ссылка: Рис.3)(Ссылка: Рис.4)(Ссылка: Рис.5)



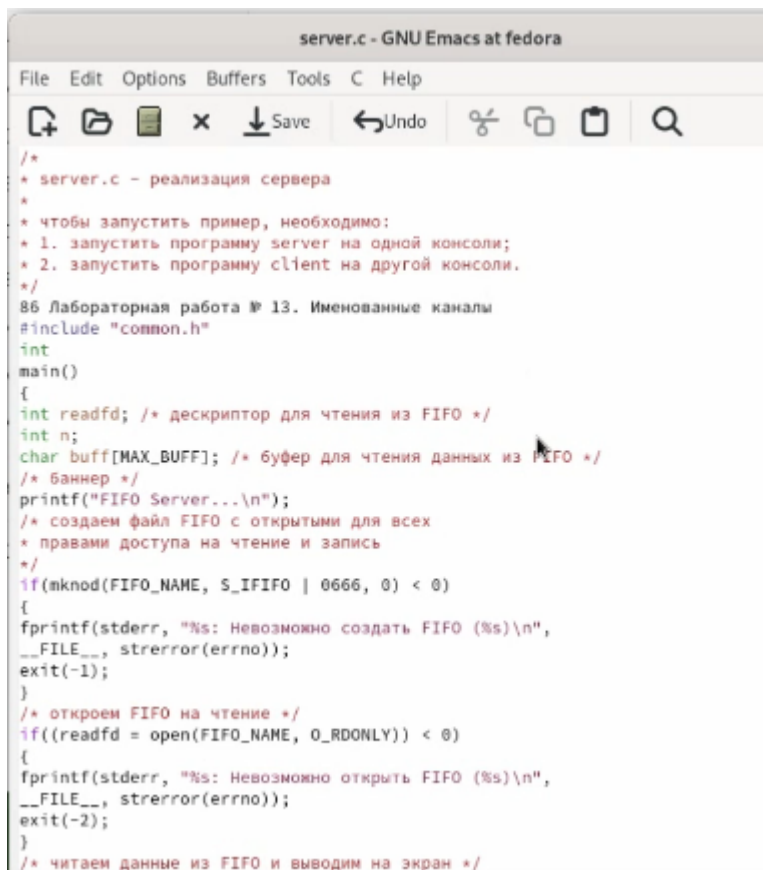
```
/*
 * common.h - заголовочный файл со стандартными определениями
 */
#ifndef __COMMON_H__
#define __COMMON_H__
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#define FIFO_NAME "/tmp/fifo"
#define MAX_BUFF 80
#endif /* __COMMON_H__ */
```

Рис. 3: Код файла common.h



```
/*
 * 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");
    /* 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. 4: Код файла client.c

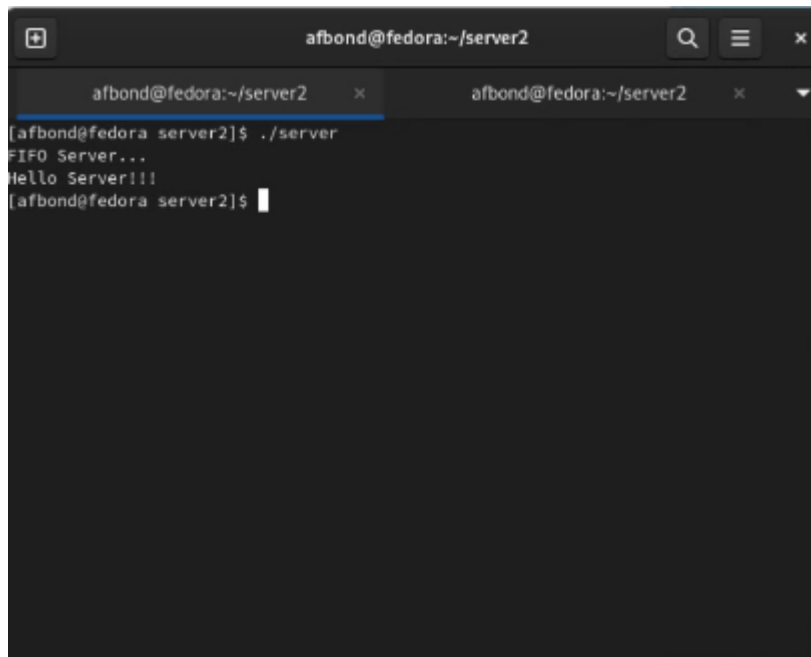


```
server.c - GNU Emacs at fedora
File Edit Options Buffers Tools C Help
[Icons: New, Open, Save, Undo, Redo, Copy, Paste, Find]

/*
 * server.c - реализация сервера
 *
 * чтобы запустить пример, необходимо:
 * 1. запустить программу server на одной консоли;
 * 2. запустить программу client на другой консоли.
 */
86 Лабораторная работа № 13. Именованные каналы
#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);
    }
    /* читаем данные из FIFO и выводим на экран */
}
```

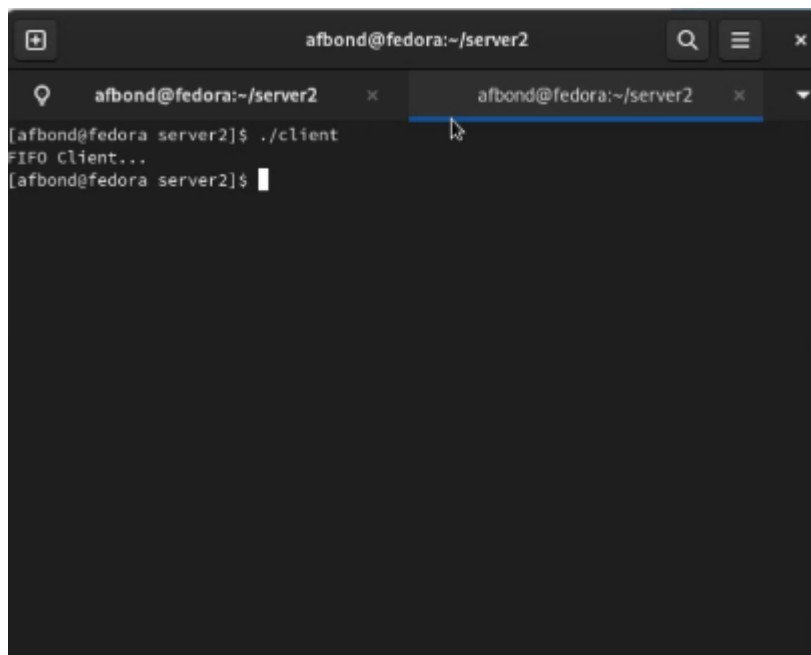
Рис. 5: Код файла server.c

После чего скомпилировал эти файлы в два исполняемых server и client. Проверил их на работоспособность, открыв каждый в отдельном терминале. (Ссылка: Рис.6)(Ссылка: Рис.7)



```
afbond@fedora:~/server2
afbond@fedora:~/server2 x afbond@fedora:~/server2 x
[afbond@fedora server2]$ ./server
FIFO Server...
Hello Server!!!
[afbond@fedora server2]$
```

Рис. 6: Работа исполняемого файла Server в терминале

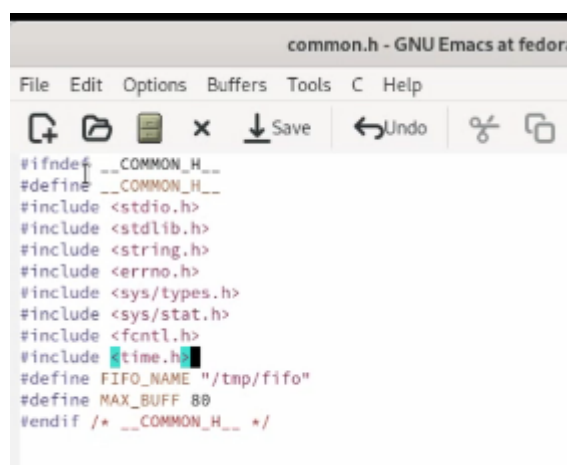


```
afbond@fedora:~/server2
afbond@fedora:~/server2 x afbond@fedora:~/server2 x
[afbond@fedora server2]$ ./client
FIFO Client...
[afbond@fedora server2]$
```

Рис. 7: Работа исполняемого файла Client в терминале

Далее в каждый из файлов внес соответствующие изменения (Ссылка:

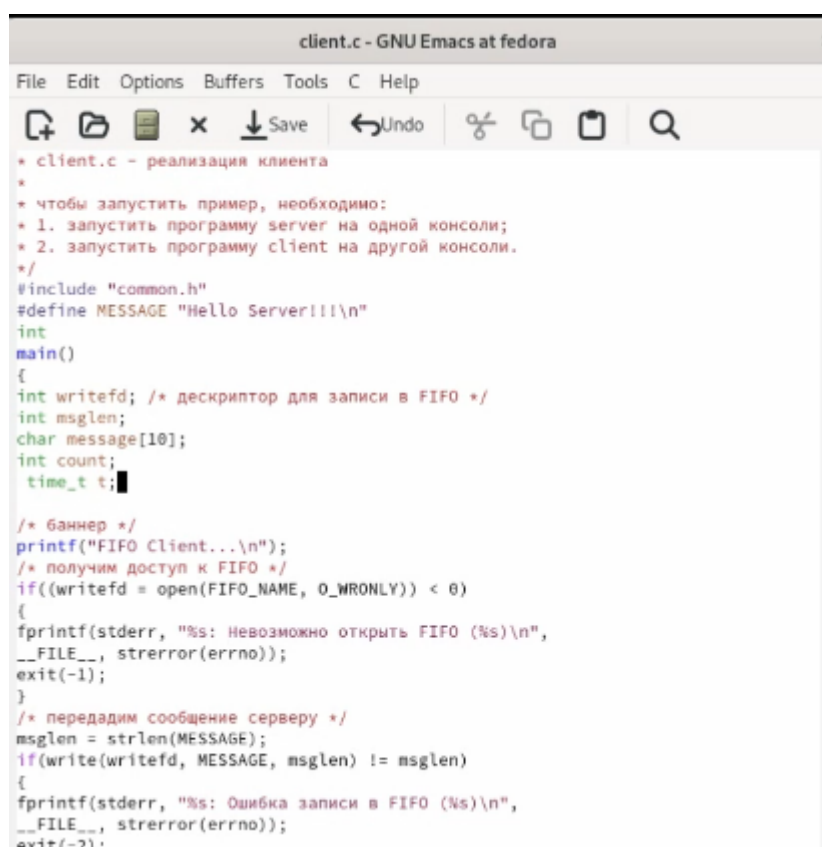
Рис.8)(Ссылка: Рис.9)(Ссылка: Рис.10)



```
common.h - GNU Emacs at fedora
File Edit Options Buffers Tools C Help
[Icons: Open, Save, Undo, Redo]

#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 <time.h>
#define FIFO_NAME "/tmp/fifo"
#define MAX_BUFF 80
#endif /* __COMMON_H__ */
```

Рис. 8: Изменение файла common.h

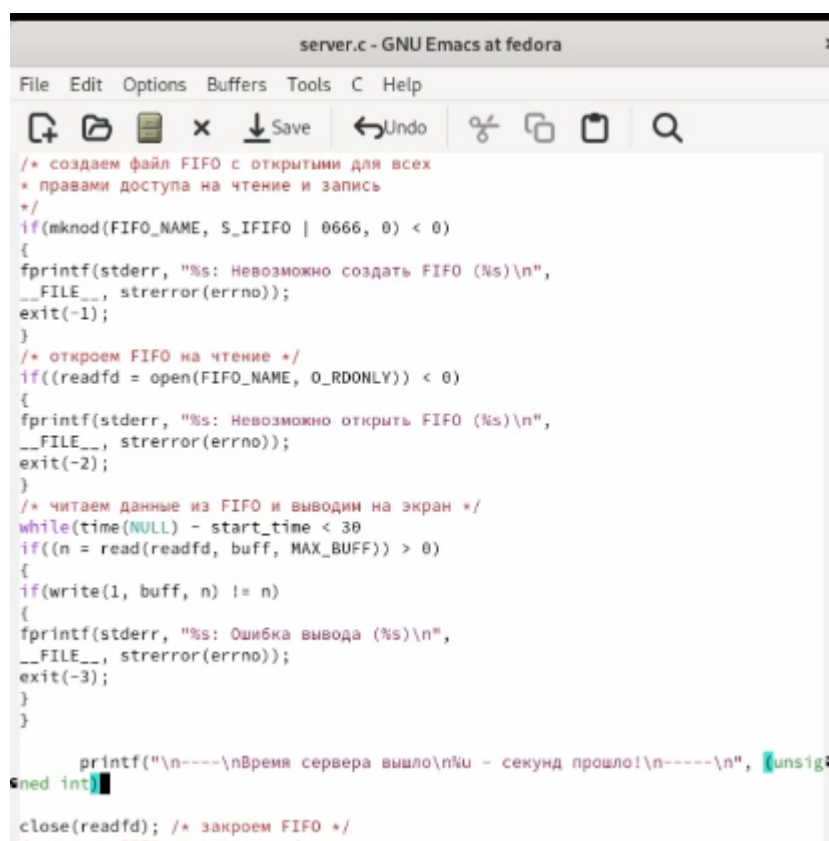


```
client.c - GNU Emacs at fedora
File Edit Options Buffers Tools C Help
[Icons: Open, Save, Undo, Redo, Search]

* 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;
    char message[10];
    int count;
    time_t t;

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

Рис. 9: Изменение файла client.c



```
server.c - GNU Emacs at fedora
File Edit Options Buffers Tools C Help
[Icons: Save, Undo, Cut, Copy, Paste, Find]

/* создаем файл FIFO с открытием 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);
}
/* читаем данные из FIFO и выводим на экран */
while(time(NULL) - start_time < 30)
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);
    }
}

printf("\n---\nВремя сервера вышло\n%u - секунд прошло!\n---\n", (unsigned
int));

close(readfd); /* закроем FIFO */
/* удалим FIFO из системы */
```

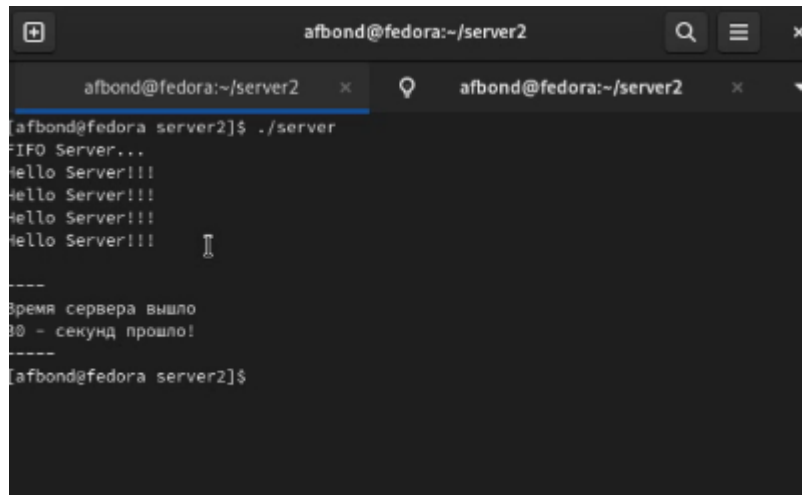
Рис. 10: Изменение файла server.c

По итогу этих изменений получилось следующее:

Работает не 1 клиент, а несколько.

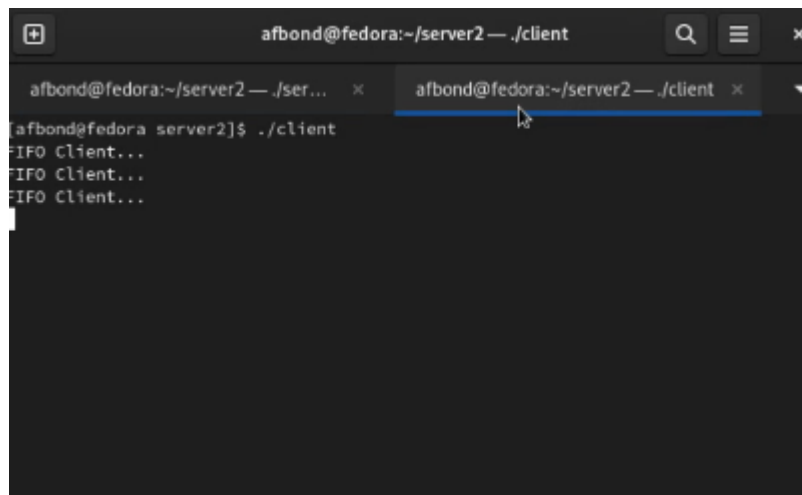
Также теперь каждый последующий клиент присоединяется с определенным интервалом.

Теперь сервер работает не бесконечно, а прекращает работу через 30 секунд и выводит соответствующее сообщение. (Ссылка: Рис.11)(Ссылка: Рис.12)



```
afbond@fedora:~/server2
afbond@fedora:~/server2$ ./server
FIFO Server...
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
-----
Время сервера вышло
30 - секунд прошло!
-----
afbond@fedora:~/server2$
```

Рис. 11: Демонстрация работы получившегося файла server в терминале



```
afbond@fedora:~/server2 — ./client
afbond@fedora:~/server2$ ./client
FIFO Client...
FIFO Client...
FIFO Client...
```

Рис. 12: Демонстрация работы получившегося файла client в терминале

Если сервер завершит работу, не закрыв канал FIFO, то клиент может зависнуть в ожидании ответа от сервера. Это происходит потому, что канал FIFO остается открытым для чтения в клиенте, и процесс не может получить конец файла (EOF), что означает, что он продолжает ожидать ответа от сервера. Кроме того, если сервер не закроет канал FIFO перед завершением, канал будет оставаться в системе, занимая ресурсы, и его необходимо будет удалить вручную.

Выводы

Таким образом, в ходе работы с именованными каналами были приобретены практические навыки создания и использования именованных каналов для организации межпроцессного взаимодействия.

Ответы на контрольные вопросы

1. В чем ключевое отличие именованных каналов от неименованных?

Именованные и неименованные каналы в UNIX/Linux используются для обмена данными между процессами.

Ключевое отличие между ними заключается в том, что именованные каналы имеют имена в файловой системе, и к ним можно обратиться из других процессов, как к обычному файлу. В то время как неименованные каналы не имеют имен в файловой системе, и могут использоваться только между процессами, которые были созданы из одного родительского процесса.

Кроме того, именованные каналы могут использоваться для обмена данными между процессами, которые не связаны друг с другом и не являются дочерними процессами родительского процесса. В то время как неименованные каналы могут использоваться только для обмена данными между процессами, которые были созданы из одного родительского процесса.

2. Возможно ли создание неименованного канала из командной строки?

Да, создание неименованного канала (трубы) из командной строки возможно при помощи оператора `|` (вертикальная черта), который позволяет направлять вывод одной команды на вход другой команды. Например, команда `ls -l | grep "file.txt"` создаст неименованный канал между командами `ls` и `grep`, где вывод команды `ls` будет передан на вход команды `grep`, которая отфильтрует строки, содержащие `"file.txt"`. Однако при использовании таких каналов нет возможности управлять процессами, передающими данные через этот канал, в отличие от именованных

каналов, которые могут быть созданы в явном виде и управляемы через файловую систему.

3. Возможно ли создание именованного канала из командной строки?

Да, создание именованного канала (также известного как FIFO) возможно из командной строки с помощью утилиты `mkfifo`. Синтаксис команды выглядит следующим образом:

```
mkfifo имя_канала
```

Здесь `имя_канала` - это имя, которое будет присвоено каналу. После создания именованного канала, его можно использовать для передачи данных между процессами.

4. Опишите функцию языка C, создающую неименованный канал.

Функция языка C, используемая для создания неименованного канала - это `pipe()`. Она определена в заголовочном файле `<unistd.h>` и имеет следующее объявление:

```
int pipe(int pipefd[2]);
```

Аргумент `pipefd` - это массив целых чисел длиной 2, который будет использоваться для возврата файловых дескрипторов, связанных с каналом. Первый элемент массива (т.е. `pipefd[0]`) будет использоваться для чтения из канала, а второй элемент (т.е. `pipefd[1]`) - для записи в канал.

5. Опишите функцию языка C, создающую именованный канал.

Функция языка C для создания именованного канала называется `mkfifo()` и объявлена в заголовочном файле `<sys/stat.h>`.

Синтаксис функции:

```
int mkfifo(const char *pathname, mode_t mode);
```


Параметры функции:

pathname - указатель на строку, содержащую имя именованного канала.

mode - режим доступа, который будет присвоен именованному каналу.

Функция возвращает 0 в случае успеха и -1 в случае ошибки.

6. Что будет в случае прочтения из fifo меньшего числа байтов, чем находится в канале? Большого числа байтов?

При чтении из FIFO будет считано только то количество байтов, которое доступно в данный момент. Если запрошенное количество байтов меньше, чем находится в канале, то функция чтения блокируется до тех пор, пока не будет доступно запрошенное количество байтов.

Если запрошенное количество байтов больше, чем находится в канале, то функция чтения блокируется до тех пор, пока не будет доступно запрошенное количество байтов. Если другой процесс записывает данные в FIFO, то функция чтения будет блокироваться, пока не будет доступно запрошенное количество байтов. Если процесс записи закрывает FIFO, то функция чтения возвращает 0, что сигнализирует о том, что канал закрыт

7. Аналогично, что будет в случае записи в fifo меньшего числа байтов, чем позволяет буфер? Большого числа байтов?

Если в FIFO записывается меньшее количество байт, чем позволяет буфер, то они будут записаны в FIFO, а оставшееся место в буфере останется свободным.

Если в FIFO записывается большее количество байт, чем позволяет буфер, то запись будет произведена только до тех пор, пока весь буфер не будет заполнен. После этого процесс, который пишет в FIFO, будет заблокирован до тех пор, пока другой процесс не прочтает данные из FIFO и не освободит место в буфере.

8. Могут ли два и более процессов читать или записывать в канал?

Да, два и более процессов могут читать или записывать в канал, если они имеют соответствующие права доступа к каналу и следуют правильному протоколу чтения/записи в канал.

При чтении из канала каждый процесс получает свою копию данных, записанных в канал. Таким образом, если два или более процесса читают из канала, они могут получить одни и те же данные в разных экземплярах.

При записи в канал каждый процесс может записывать данные в канал независимо от других процессов. Тем не менее, при записи в канал следует учитывать размер буфера канала, чтобы избежать блокировки процессов, ожидающих чтения данных из канала.

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` используется для записи данных в файловый дескриптор. Она возвращает количество записанных байтов или `-1` в случае ошибки.

Синтаксис функции:

```
ssize_t write(int fd, const void *buf, size_t count);
```

где:

`fd` - файловый дескриптор, куда будут записываться данные;

`buf` - указатель на буфер с данными, которые нужно записать;

`count` - количество байтов для записи.

В программе `server.c` значение `1` (единица) в вызове функции `write` указывает на файловый дескриптор стандартного вывода (`STDOUT_FILENO`), куда будут выводиться данные из канала. В данном случае, функция `write` используется для вывода прочитанных данных из FIFO на экран.

10. Опишите функцию `strerror`.

`strerror()` - это функция языка программирования C, которая преобразует код ошибки в строку сообщения об ошибке. Она возвращает указатель на строку, содержащую сообщение об ошибке, соответствующее переданному коду ошибки.