

# **Лабораторная работа №15**

**Дисциплина: Операционные системы**

Галиев Казиз Жарылкасымович

# Содержание

Цель работы	5
Выполнение лабораторной работы	6
Выводы	10
Контрольные вопросы	11
Список литературы	14

## Список иллюстраций

0.1	Файл common.h . . . . .	6
0.2	Файл Makefile . . . . .	6
0.3	Файл client.c . . . . .	7
0.4	Файл server.c . . . . .	8
0.5	Продолжение файла server.c . . . . .	8
0.6	Выполнение утилиты make . . . . .	9
0.7	Выполнение программы на двух терминалах . . . . .	9

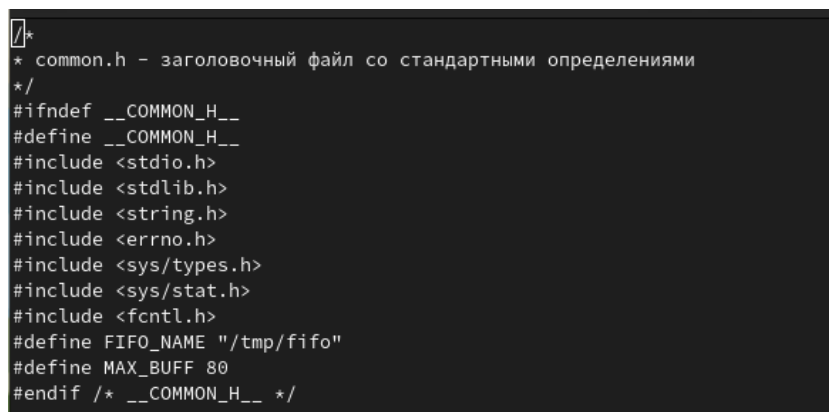
## Список таблиц

## **Цель работы**

Приобретение практических навыков работы с именованными каналами.

# Выполнение лабораторной работы

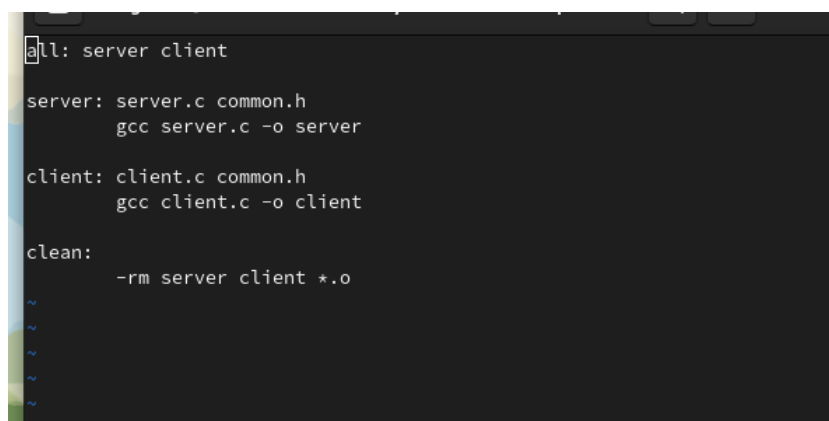
1. Создадим файл common.h (рис. @fig:001).



```
*
* 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. 0.1: Файл common.h

2. Создадим файл Makefile (рис. @fig:002).



```
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. 0.2: Файл Makefile

3. Создадим файл client.c (рис. @fig:003).

```
/*
 * 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. 0.3: Файл client.c

4. Внесем изменения в файл server.c (рис. @fig:004), (рис. @fig:005).

```

#include "time.h"
#include "common.h"
void display() {
    printf("/n Server timeout...%u seconds passed:\n 30 seconds:\n",clock());
}
int main()
{
    clock_t start, now;
    start = time(NULL);
    int readfd; /* дескриптор для чтения из FIFO */
    int n;
    char buff[MAX_BUFF]; /* буфер для чтения данных из FIFO */
    /* баннер */
    printf("Hell Server...\n\n");
    //if (mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)
    //{
    //printf(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);
    }
    for(;;)
    {
        while ((n = read(readfd, buff, MAX_BUFF)) > 0)
        {
            if (write(1, buff, n) != n)
            {
                fprintf(stderr, "%s: Ошибка вывода (%s)\n",
                __FILE__, strerror(errno));
            }
            sleep(5);
        }
        now=time(NULL);
        if (now-start>30)
        {
            display();
            return 0;
        }
    }
}

```

Рис. 0.4: Файл server.c

```

}
}
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. 0.5: Продолжение файла server.c

5. Выполнение утилиты make (рис. @fig:006).



```

[kzgaliev@fedora 11]$ make
gcc server.c -o server
server.c: В функции «main»:
server.c:30:13: предупреждение: неявная декларация функции «read»; имелось в виду «fread»? [-Wimplicit-function-declaration]
   30 | while ((n = read(readfd, buff, MAX_BUFF)) > 0)
      |             ^~~~~
      |             fread
server.c:32:5: предупреждение: неявная декларация функции «write»; имелось в виду «fwrite»? [-Wimplicit-function-declaration]
   32 | if (write(1, buff, n) != n)
      |     ^~~~~
      |     fwrite
server.c:37:1: предупреждение: неявная декларация функции «sleep» [-Wimplicit-function-declaration]
   37 | sleep(5);
      | ^~~~~
server.c:46:1: предупреждение: неявная декларация функции «close»; имелось в виду «pclose»? [-Wimplicit-function-declaration]
   46 | close(readfd); /* закроем FIFO */
      | ^~~~~
      | pclose
server.c:48:5: предупреждение: неявная декларация функции «unlink» [-Wimplicit-function-declaration]
   48 | if (unlink(FIFO_NAME) < 0)
      |     ^~~~~
gcc client.c -o client
client.c: В функции «main»:
client.c:26:4: предупреждение: неявная декларация функции «write»; имелось в виду «fwrite»? [-Wimplicit-function-declaration]
   26 | if (write(writefd, MESSAGE, msglen) != msglen)
      |     ^~~~~
      |     fwrite
client.c:33:1: предупреждение: неявная декларация функции «close»; имелось в виду «pclose»? [-Wimplicit-function-declaration]
   33 | close(writefd);
      | ^~~~~
      | pclose
[kzgaliev@fedora 11]$

```

Рис. 0.6: Выполнение утилиты make

6. Выполнение программы на двух терминалах (рис. @fig:007).

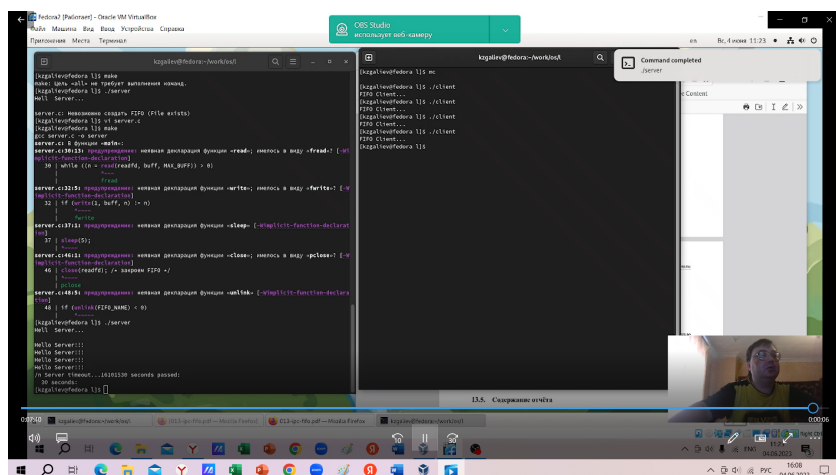


Рис. 0.7: Выполнение программы на двух терминалах

## **Выводы**

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

# Контрольные вопросы

## 1. В чем ключевое отличие именованных каналов от неименованных?

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).

## 2. Возможно ли создание неименованного канала из командной строки?

Для создания неименованного канала используется системный вызов `pipe`. Массив из двух целых чисел является выходным параметром этого системного вызова. Если вызов выполнен нормально, то массив содержит два файловых дескриптора: для чтения информации из канала и для записи в него соответственно. Когда процесс порождает другой процесс, дескрипторы родителя наследуются дочерним процессом, и, таким образом, осуществляется связь между двумя процессами. Один из них 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, то родительский процесс создает два канала, один из которых используется для передачи данных в одну сторону, а другой — в другую.

### 3. Возможно ли создание именованного канала из командной строки?

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 р)` 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`.

### 4. Опишите функцию языка C, создающую неименованный канал.

Неименованный канал создается вызовом `pipe`, который заносит в массив `int [2]` два дескриптора открытых файлов. `fd[0]` – открыт на чтение, `fd[1]` – на запись (вспомните `STDIN == 0, STDOUT == 1`). Канал уничтожается, когда будут закрыты все файловые дескрипторы, ссылающиеся на него.

### 5. Опишите функцию языка C, создающую именованный канал.

Именованный канал FIFO доступен как объект в файловой системе. При этом, до открытия объекта FIFO на чтение, собственно коммуникационного объекта не создаётся. После открытия объекта FIFO в одном процессе на чтение, а в другом на запись, возникает ситуация полностью эквивалентная использованию неименованного канала. Объект FIFO в файловой системе создаётся вызовом функции `int mkfifo(const char *pathname, mode_t mode);` Основное отличие между `pipe` и FIFO – то, что `pipe` могут совместно использовать только процессы, находящиеся в отношении родительский-дочерний, а FIFO может использовать любая пара процессов.

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

При чтении числа байт, меньшего чем находится в канале, возвращается требуемое число байтов, остаток сохраняется для последующих чтений. При чтении числа байт, большего чем находится в канале, возвращается доступное число байт.

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

Запись числа байт меньше чем `PIPE_BUF` выполняется атомарно. При записи из нескольких процессов данные не перемешиваются. При записи числа байт больше чем `PIPE_BUF` атомарность операции не гарантируется.

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

С точки зрения процессов, канал выглядит как пара открытых файловых дескрипторов – один на чтение и один на запись (можно больше, но неудобно). Мы можем писать в канал до тех пор, пока есть место в буфере, если место в буфере кончится – процесс будет заблокирован на записи.

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)?

Для записи информации в канал используется системный вызов `write`. Для чтения информации из канала — системный вызов `read`. Первый аргумент этих вызовов — дескриптор канала, имеющий тип `int`, второй — указатель на область памяти, с которой происходит обмен, имеет тип `void`, третий — количество байт, целочисленный тип. Оба вызова возвращают число переданных байт (либо значение «-1» при ошибке). При завершении использования канала процесс выполняет системный вызов `close`.

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, понятное человеку.

## **Список литературы**