

# **Отчёт по лабораторной работе №14**

**Дисциплина: Операционные системы**

Батова Ирина Сергеевна, НММбд-01-22

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
<b>4</b>	<b>Выводы</b>	<b>13</b>
<b>5</b>	<b>Контрольные вопросы</b>	<b>14</b>

## Список иллюстраций

3.1	Создание файлов для работы . . . . .	7
3.2	Редактирование файла common.h . . . . .	7
3.3	Редактирование файла server.c . . . . .	8
3.4	Редактирование файла server.c . . . . .	9
3.5	Редактирование файла client.c . . . . .	10
3.6	Редактирование файла client.c . . . . .	10
3.7	Компиляция исполняемых файлов . . . . .	11
3.8	Запуск программы . . . . .	11
3.9	Проверка завершения работы сервера при незакрытом канале . .	12

## **Список таблиц**

# **1 Цель работы**

Приобретение практических навыков работы с именованными каналами.

## 2 Задание

Изучите 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

Для начала работы создаем командой 'touch' четыре файла - common.h, server.c, client.c и Makefile (рис. 3.1).

```
[isbatova@fedora ~]$ touch common.h server.c client.c Makefile  
[isbatova@fedora ~]$
```

Рис. 3.1: Создание файлов для работы

Для начала открываем в редакторе файл 'common.h'. Для корректной работы других файлов добавляем к листингу из лабораторной работы два заголовочных файла -unistd.h и time.h (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: Редактирование файла common.h

Далее открываем файл 'server.c'. Нам нужно, чтобы сервер заканчивал работу через 30 секунд. Для реализации данного действия сначала обозначаем время начало работы (clock\_t start=time(NULL)), а затем вносим прочтение данных из FIFO и вывод их на экран под цикл while, который работает только пока разница между текущим временем и временем начала работы меньше 30 секунд (рис. 3.3, 3.4).

```
/*
 * 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 */

    /* 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);
    }
}
```

Рис. 3.3: Редактирование файла server.c



```

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

```

Рис. 3.4: Редактирование файла server.c

Далее открываем файл 'client.c'. Нам нужно, чтобы клиенты передавали текущее сообщение раз в пять секунд. Для этого добавляем цикл for, который анализирует количество и отправляет сообщения о текущем времени, добавляем команды для генерации этих сообщений (long int ttime=time(NULL) и char\* text=ctime(&ttime)), а также команду sleep(5), которая останавливает работу клиента на 5 секунд (рис. 3.5, 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"

#define MESSAGE "Hello Server!!!\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);
            break;
        }
    }

```

Рис. 3.5: Редактирование файла client.c

```

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

        /* 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);
        }

        sleep(5);
    }

    /* закроем доступ к FIFO */
    close(writefd);

    exit(0);
}

```

Рис. 3.6: Редактирование файла client.c

В Makefile вводим листинг, соответствующий лабораторной работе, и оставляем

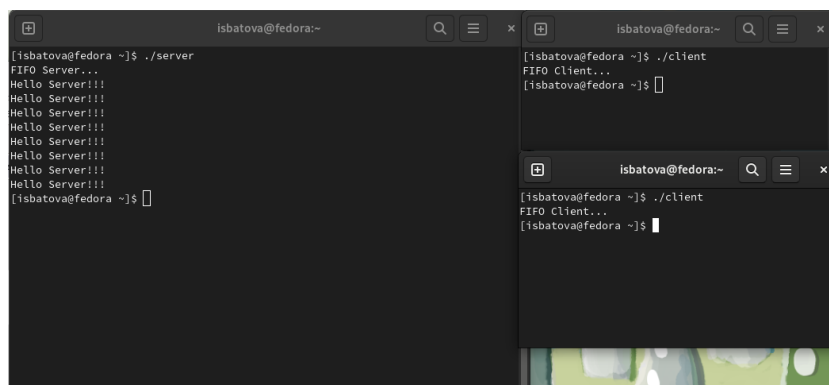
без изменений.

Далее вводим команды 'make server' и 'make client' для компиляции исполняемых файлов (рис. 3.7).

```
[isbatova@fedora ~]$ make server
gcc server.c -o server
[isbatova@fedora ~]$ make client
gcc client.c -o client
[isbatova@fedora ~]$
```

Рис. 3.7: Компиляция исполняемых файлов

После этого проверяем работу наших скриптов: открываем три окна терминала, в одном запускаем файл server, в двух других - файл client. Каждый "клиент" вывел по четыре сообщения, а спустя тридцать секунд сервер завершил работу. Скрипты работают корректно (рис. 3.8).



The image shows three terminal windows. The leftmost window runs the server program, which prints 'FIFO Server...' and eight 'Hello Server!!!' messages before exiting. The two windows on the right run the client program, each printing 'FIFO Client...' and waiting for input. The terminal windows are titled 'isbatova@fedora:~' and have search and menu icons in the title bar.

Рис. 3.8: Запуск программы

Для проверки, что будет в случае, если сервер завершит работу, не закрыв канал, запускаем файл server и завершаем его раньше. При попытке снова запустить сервер, программа выдает ошибку о невозможности создания FIFO, так как уже создан один канал.

```
[isbatova@fedora ~]$ ./server
FIFO Server...
server.c: Невозможно создать FIFO (File exists)
[isbatova@fedora ~]$
```

Рис. 3.9: Проверка завершения работы сервера при незакрытом канале

## **4 Выводы**

В ходе данной лабораторной работы я приобрела практические навыки работы с именованными каналами.

## 5 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. Для создания неименованного канала из командной строки используется символ | для объединения нескольких процессов.
3. Для создания именованного канала из командной строки используется две команды - mkhod и mkfifo.
4. Функция языка C, создающая неименованный канал - `int pipe(int fd[2]);`. При нормальном выполнении вызова массив, являющийся выходным параметром этого системного вызова, содержит два файловых дескриптора - `fd[0]`, дескриптор для чтения из канала, и `fd[1]`, дескриптор для записи в канал.
5. Функции языка C, создающие именованный канал:
  - `int mkfifo(const char *pathname, mode_t mode);`. Параметры последовательно отвечают за путь расположения FIFO и режим работы с FIFO.
  - `int mkhod(const char *pathname, mode_t mode);`. Параметры последовательно отвечают за путь расположения FIFO и режим работы с FIFO.

- “mkhod (namefile, IFIFO | 0666, 0”. Параметр “namefile” отвечает за имя канала, цифры - за разрешения доступа на запись и на чтение любому запросившему процесс.
6. В случае прочтения меньшего числа байтов, чем находится в канале, возвращается требуемое число, а остаток сохраняется для следующих прочтений. В случае прочтения большего числа байтов возвращается только доступное число.
  7. В случае записи в FIFO меньшего числа байтов, чем позволяет буфер, несколько процессов одновременно записываются в канал и порции данных от этих процессов не перемешиваются. В случае записи в FIFO большего числа байтов, чем позволяет буфер, не гарантируется не перемешивание порции данных от нескольких процессов, а вызов write блокируется до освобождения нужного количества байтов.
  8. Количество процессов, которые читают или записывают в канал, не ограничено.
  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, при этом возвращаемое значение должно быть положительным, но меньше числа count. Единица в вызове этой функции в server.c означает идентификатор стандартного потока вывода.
  10. Функция strerror интерпретирует номер ошибки, передаваемой в функцию в качестве аргумента errno, в текстовое сообщение (строку). Ошибки возникают при вызове функций стандартных Си-библиотек. Возвращен-

ный 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 в зависимости от платформы и компилятора.