

# Отчёт по лабораторной работе №14

## Дисциплина: Операционные системы

Елизавета Андреевна Алмазова

### Содержание

Цель работы .....	1
Задание .....	1
Теоретическое введение .....	1
Выполнение лабораторной работы .....	3
Выводы .....	6
Ответы на контрольные вопросы .....	6

### Цель работы

Цель данной лабораторной работы - приобретение практических навыков работы с именованными каналами.

### Задание

Изучите приведённые в тексте программы `server.c` и `client.c`. Взяв 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?

### Теоретическое введение

Одним из видов взаимодействия между процессами в операционных системах является обмен сообщениями. Под сообщением понимается последовательность байтов, передаваемая от одного процесса другому.

В операционных системах типа UNIX есть 3 вида межпроцессорных взаимодействий: общеюниксные (именованные каналы, сигналы), System V Interface Definition (SVID —

разделяемая память, очередь сообщений, семафоры) и BSD (сокеты). 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. Файлы именованных каналов создаются функцией `mkfifo`.

Первый параметр — имя файла, идентифицирующего канал, второй параметр — маска прав доступа к файлу. После создания файла канала процессы, участвующие в обмене данными, должны открыть этот файл либо для записи, либо для чтения. При закрытии файла сам канал продолжает существовать. Для того чтобы закрыть сам канал, нужно удалить его файл, например с помощью вызова `unlink(2)`.

Рассмотрим работу именованного канала на примере системы клиент–сервер. Сервер создаёт канал, читает из него текст, посылаемый клиентом, и выводит его на терминал. Вызов функции `mkfifo()` создаёт файл канала (с именем, заданным макросом `FIFO_NAME`): `mkfifo(FIFO_NAME, 0600)`.

В качестве маски доступа используется восьмеричное значение `0600`, разрешающее процессу с аналогичными реквизитами пользователя чтение и запись. Можно также установить права доступа `0666`. Открываем созданный файл для чтения: `f = fopen(FIFO_NAME, O_RDONLY)`. Ждём сообщение от клиента. Сообщение читаем с помощью функции `read()` и печатаем на экран. После этого удаляется файл `FIFO_NAME` и сервер прекращает работу.

Клиент открывает FIFO для записи как обычный файл: `f = fopen(FIFO_NAME, O_WRONLY)`. Посылаем сообщение серверу с помощью функции `write()`. Для создания файла FIFO можно использовать более общую функцию `mknod(2)`, предназначенную для создания специальных файлов различных типов (FIFO, сокеты, файлы устройств и обычные файлы для хранения данных).

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
```

```
#include <fcntl.h>
```

```
#include <unistd.h>
```

```
int mknod(const char *pathname, mode_t mode, dev_t dev);
```

Тогда, вместо `mkfifo(FIFO_NAME, 0600)` пишем `mknod(FIFO_NAME, S_IFIFO | 0600, 0)`.

Каналы представляют собой простое и удобное средство передачи данных, которое, однако, подходит не во всех ситуациях. Например, с помощью каналов довольно трудно организовать обмен асинхронными сообщениями между процессами.

## Выполнение лабораторной работы

1. Изучила приведённые в тексте программы server.c и client.c. Создала файлы common.h, server.c, client.c, Makefile и, взяв примеры за образец, написала аналогичные программы, внося следующие изменения (рис.1, рис.2, рис.3, рис.4):
  1. Работает не 1 клиент, а несколько (например, два).
  2. Клиенты передают текущее время раз в пять секунд. Для получения текущего времени в файле client.c я использовала функции библиотеки time.h, добавленную в файле common.h, а для приостановки работы клиента функцию sleep(5). Это время передается три раза с помощью цикла.
  3. Я изменила файл server.c. Сервер работает не бесконечно, а прекращает работу через 30 сек. Использовала функцию clock() для определения времени работы сервера. Если сервер завершит работу, не закрыв канал, при новом включении сервера появится ошибка при создании канала, так как один уже существует.

```
1 /*
2  * client.c - реализация клиента
3  *
4  * чтобы запустить пример, необходимо:
5  * 1. запустить программу server на одной консоли;
6  * 2. запустить программу client на другой консоли.
7  */
8
9 #include "common.h"
10
11 #define MESSAGE "Hello Server!!!\n"
12
13 int main(){
14     int writefd; /* дескриптор для записи в FIFO */
15     int msglen;
16
17     /* баннер */
18     printf("FIFO Client...\n");
19
20     /* цикл передачи сообщений */
21
22     for (int i=0; i<3; i++) {
23         /* получим доступ к FIFO */
24         if((writefd = open(FIFO_NAME, O_WRONLY)) < 0){
25             fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n", __FILE__, strerror(errno));
26             exit(-1);
27         }
28
29         /* получим текущее время */
30         long int timer=time(0);
31         char* timeprint=ctime(&timer);
32
33         /* передадим сообщение серверу */
34         msglen = strlen(timeprint);
35         if(write(writefd, timeprint, msglen) != msglen) {
36             fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n");
37             exit(-2);
38         }
39
40         /* приостановка работы */
41
42         sleep(5);
43     }
44
45     /* закроем доступ к FIFO */
46     close(writefd);
47
48     exit(0);
49 }
```

Рисунок 1 - client.c.

```
/*
 * 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 <time.h>

#define FIFO_NAME "/tmp/fifo"
#define MAX_BUFF 80

#endif /* __COMMON_H__ */
```

*Рисунок 2 - common.h.*

```

1 /*
2  * server.c - реализация сервера
3  *
4  * чтобы запустить пример, необходимо:
5  * 1. запустить программу server на одной консоли;
6  * 2. запустить программу client на другой консоли.
7  */
8
9 #include "common.h"
10
11 int main()
12 {
13     int readfd; /* дескриптор для чтения из FIFO */
14     int n;
15     char buff[MAX_BUFF]; /* буфер для чтения данных из FIFO */
16
17     /* баннер */
18     printf("FIFO Server...\n");
19
20     /* создаем файл FIFO с открытыми для всех
21      * правами доступа на чтение и запись
22      */
23     if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0) {
24         fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n", __FILE__, strerror(errno));
25         exit(-1);
26     }
27
28     /* откроем FIFO на чтение */
29     if((readfd = open(FIFO_NAME, O_RDONLY)) < 0){
30         fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n", __FILE__, strerror(errno));
31         exit(-2);
32     }
33
34     /* отсчет времени работы */
35
36     clock_t count = clock();
37
38     while(clock()-count < 30) {
39         /* читаем данные из FIFO и выводим на экран */
40         while((n=read(readfd,buff,MAX_BUFF)) > 0) {
41             if(write(1,buff,n)!=n) {
42                 printf(stderr, "%s: Ошибка вывода (%s)\n", __FILE__, strerror(errno));
43                 exit(-3);
44             }
45         }
46     }
47
48     close(readfd); /* закроем FIFO */
49
50     /* Удалим FIFO из системы */
51     if(unlink(FIFO_NAME) < 0) {
52         fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n", __FILE__, strerror(errno));
53         exit(-4);
54     }
55     exit(0);
56 }

```

Рисунок 3 - server.c.

```

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

```

Рисунок 4 - Makefile.

2. Программы работают верно (рис.5, рис.6).







 client	25.5 kB	14:44
 client.c	1.3 kB	14:41
 common.h	395 bytes	14:42
 Makefile	152 bytes	14:42
 server	25.6 kB	14:44
 server.c	1.7 kB	14:44

Рисунок 5 - Созданные с помощью `make all` файлы.

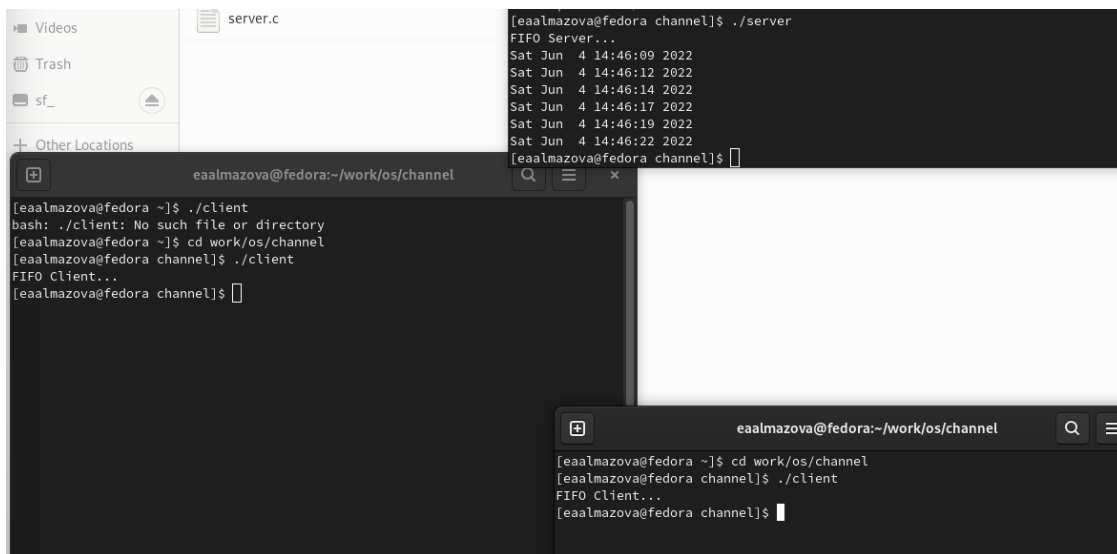


Рисунок 6 - Одновременная работа двух клиентов.

## Выводы

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

## Ответы на контрольные вопросы

1. В чем ключевое отличие именованных каналов от неименованных?

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. Возможно ли создание неименованного канала из командной строки?

Да, возможно с помощью символа `|`.

3. Возможно ли создание именованного канала из командной строки?

Да, возможно с помощью команды `mknod/mkfifo`.

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.

В рамках одного процесса `pipe` смысла не имеет, передать информацию о нем в произвольный процесс нельзя (имени нет, а номера файловых дескрипторов в каждом процессе свои). Единственный способ использовать `pipe` – унаследовать дескрипторы `pipe` при вызове `fork` (и последующем `exec`). После вызова `fork` канал окажется открытым на чтение и запись в родительском и дочернем процессе. Т.е. теперь на него будут ссылаться 4 дескриптора. Теперь надо определиться с направлением передачи данных – если надо передавать данные от родителя к потомку, то родитель закрывает дескриптор на чтение, а потомок – дескриптор на запись.

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)`. Команда `mkfifo` позволяет задействовать одноименную утилиту, предназначенную для создания именованных программных каналов. Программные каналы предназначены для обмена данными между приложениями (или в рамках одного приложения) и представляют собой буферы в памяти, поддерживающие операции чтения и записи с блокировками.

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?

В случае прочтения меньше числа байтов, остаток сохраняется для последующих чтений. При прочтении большего числа, возвращается доступное число байтов, а читающий процесс должен обработать ситуацию, когда прочитано меньше, чем задано.

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

Запись меньшего числа байтов атомарна – если несколько процессов записывают в канал, порции данных не перемешиваются. При записи больше числа байтов, вызов `write(2)` блокируются до освобождения, атомарность не гарантируется. Если процесс записывает данные в неоткрытый на чтение канал, то генерируется сигнал `SIGPIPE`, а вызов `write(2)` возвращает 0 с установкой ошибки. Если обработка сигнала `SIGPIPE` не установлена, то обрабатывается по умолчанию – процесс закрывается.

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

Могут, однако может случиться так, что будут прочитана или записана только часть данных.

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 данных в файл используется системный вызов `write()`: `ssize_t write (int fd, const void * buffer, size_t count)`; В принципе `write()` выполняет процедуру, обратную `read()`: записывает `count` байтов из буфера `buffer` в файл с дескриптором `fd`, возвращая количество записанных байтов или -1 в случае ошибки. 1 - это идентификатор (дескриптор потока) стандартного потока вывода.

10. Опишите функцию `strerror`.

Функция `strerror()` возвращает строку, описывающую код ошибки, переданный в аргументе `errnum`, возможно с учетом категории `LC_MESSAGES` текущей локали для выбора соответствующего языка. Приложение не должно изменять строку. Строка может измениться при последующем вызове `perror()` или `strerror()`. В библиотеке нет функций изменяющих эту строку.

Функция `strerror()` возвращает соответствующее описание ошибки или сообщение о том, что ошибка неизвестна. Значение `errno` при удачном вызове не меняется, а при ошибке устанавливается в ненулевое значение.