

---

# Отчёт по лабораторной работе №15

## "Именованные каналы"

author: Малащенко Марина Владимировна

## Цель работы:

Приобрести практические навыки работы с именованными каналами.

## 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: общедоступные (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(3)`.

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

#include <sys/stat.h>

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

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

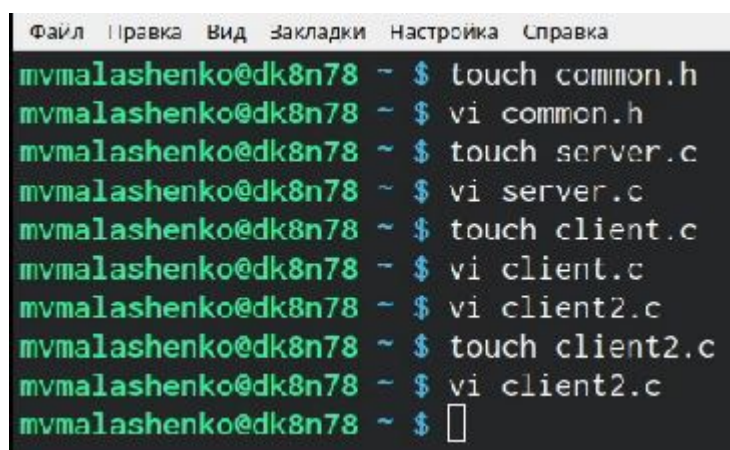
```
mknod(FIFO_NAME, S_IFIFO | 0600, 0);
```

*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.*

# Ход работы

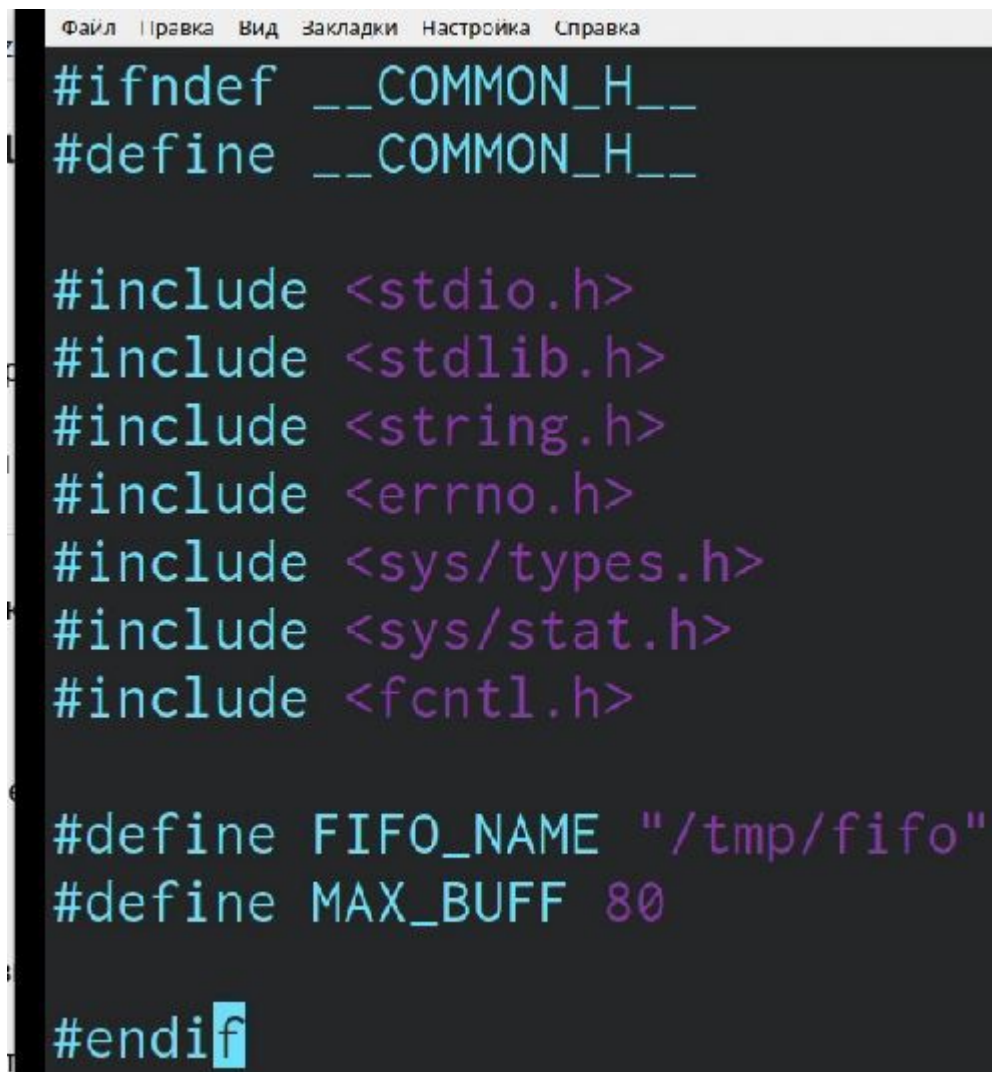
1. Изучили приведённые в тексте программы `server.c` и `client.c`. Взяв данные примеры за образец, написали аналогичные программы, внося следующие изменения:

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.



```
Файл  Правка  Вид  Закладки  Настройка  Справка
mvmalashenko@dk8n78 ~ $ touch common.h
mvmalashenko@dk8n78 ~ $ vi common.h
mvmalashenko@dk8n78 ~ $ touch server.c
mvmalashenko@dk8n78 ~ $ vi server.c
mvmalashenko@dk8n78 ~ $ touch client.c
mvmalashenko@dk8n78 ~ $ vi client.c
mvmalashenko@dk8n78 ~ $ vi client2.c
mvmalashenko@dk8n78 ~ $ touch client2.c
mvmalashenko@dk8n78 ~ $ vi client2.c
mvmalashenko@dk8n78 ~ $
```

рис.1 Терминал



```
Файл  Правка  Вид  Закладки  Настройка  Справка

#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
```

рис.2 Файл common.h

```
}

if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
{
    fprintf(stderr, "%s: Невозможно открыть FIFO (%s)
        __FILE__, strerror(errno));
    exit(-2);
}

clock_t now=time(NULL), start=time(NULL);
while(now-start<30)
{
    while((n = read(readfd, buff, MAX_BUFF)) > 0)
    {
        if(write(1, buff, n) != n)
        {
            fprintf(stderr, "%s: Ошибка вывода
                __FILE__, strerror(errno));
        }
    }
    now=time(NULL);
}
printf("server timeout, %li - seconds passed\n", (now-start));
close(readfd);

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

рис.3 Файл server.c

```
Файл  Правка  Вид  Закладки  Настройка  Справка
#include "common.h"
#define MESSAGE "Hello Server!!!\n"

int
main()
{
    int msg, len, i;
    long int t;

    for (i=0; i<20; i++)
    {
        sleep(3);
        t=time(NULL);
        printf("FIFO Client...\n");

        if((msg = open(FIFO_NAME, O_WRONLY)) < 0)
        {
            fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
                __FILE__, strerror(errno));
            exit(-1);
        }

        len = strlen(MESSAGE);

        if(write(msg, MESSAGE, len) != len)
        {
            fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
                __FILE__, strerror(errno));
            exit(-2);
        }

        close(msg);
    }

    exit(0);
}
```

рис.4 Файл client.c

```
Файл  Правка  Вид  Закладки  Настройка  Справка
#include "common.h"
#define MESSAGE "Hello Server!!!\n"

int
main()
{
    int writefd;
    int msglen;
    int count;
    long long int t;
    char message[10];

    for (count=0; count<=5; ++count)
    {
        sleep(5);
        t=(long long int) time(0);
        sprintf(message, "%lli", t);
        if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
        {
            fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
                    __FILE__, strerror(errno));
            exit(-1);
        }

        msglen = strlen(MESSAGE);
        if(write(writefd, MESSAGE, msglen) != msglen)
        {
            fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
                    __FILE__, strerror(errno));
            exit(-2);
        }
    }

    close(writefd);
    exit(0);
}
```

рис.5 Файл client2.c

---

## 2. Создадим Makefile:

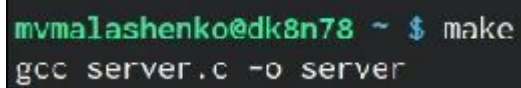
```
makefile      [----]  8 L:[  1+ 9  10/ 11] *(134 / 156b) 0045 0x02D
all: server client

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.6 Файл makefile



```
mvmalashenko@dk8n78 ~ $ make  
gcc server.c -o server
```

рис.7 Команда make (gss1)



```
gcc client.c -o client
```

рис.8 Команда make (gss2)

---

**3.** Приступим к запуску канала:

Откроем второй терминал



```
Приложения  Места  Konsole

~: bash — Konsole

Файл  Правка  Вид  Закладки  Настройка  Справка

server.c:28:7: предупреждение: неявная декларация функции «write»; имелось в в
rite»? [-Wimplicit-function-declaration]
  28 |     while((n = read(readfd, buff, MAX_BUFF)) > 0)
      |                  ^~~~~~
      |                  fread
server.c:30:7: предупреждение: неявная декларация функции «write»; имелось в в
rite»? [-Wimplicit-function-declaration]
  30 |     if(write(1, buff, n) != n)
      |        ^~~~~~
      |        fwrite
server.c:39:2: предупреждение: неявная декларация функции «close»; имелось в в
lose»? [-Wimplicit-function-declaration]
  39 |     close(readfd);
      |     ^~~~~~
      |     pclose
server.c:41:5: предупреждение: неявная декларация функции «unlink» [-Wimplicit
on-declaration]
  41 |     if(unlink(FIFO_NAME) < 0)
      |        ^~~~~~
gcc client.c -o client
client.c: В функции «main»:
client.c:13:3: предупреждение: неявная декларация функции «sleep» [-Wimplicit-
n-declaration]
  13 |     sleep(3);
      |     ^~~~~~
client.c:14:5: предупреждение: неявная декларация функции «time» [-Wimplicit-f
-declaration]
  14 |     t=time(NULL);
      |        ^~~~~~
client.c:26:6: предупреждение: неявная декларация функции «write»; имелось в в
rite»? [-Wimplicit-function-declaration]
  26 |     if(write(msg, MESSAGE, len) != len)
      |        ^~~~~~
      |        fwrite
client.c:33:3: предупреждение: неявная декларация функции «close»; имелось в в
lose»? [-Wimplicit-function-declaration]
  33 |     close(msg);
      |     ^~~~~~
      |     pclose
mvmalashenko@dk8n78 ~ $
```

рис.9 Второй терминал

Запустим в первом терминале скрипт сервера, а во втором - скрипт клиента

```
Приложения  Места  Konsole

~: bash — Konsole

Файл  Правка  Вид  Закладки  Настройка  Справка

server.c:41:5: предупреждение: неявная декларация функции «unlink» [-Wimplicit-function-declaration]
  41 |     if(unlink(FIFO_NAME) < 0)
      |         ~~~~~
gcc client.c -o client
client.c: В функции «main»:
client.c:13:3: предупреждение: неявная декларация функции «sleep» [-Wimplicit-function-declaration]
  13 |     sleep(3);
      |     ~~~~~
client.c:14:5: предупреждение: неявная декларация функции «time» [-Wimplicit-function-declaration]
  14 |     t=time(NULL);
      |     ~~~~~
client.c:26:6: предупреждение: неявная декларация функции «write»; имелось в виду «write»? [-Wimplicit-function-declaration]
  26 |     if(write(msg, MESSAGE, len) != len)
      |         ~~~~~
      |         fwrite
client.c:33:3: предупреждение: неявная декларация функции «close»; имелось в виду «close»? [-Wimplicit-function-declaration]
  33 |     close(msg);
      |     ~~~~~
      |     pclose
mvmalashenko@dk8n78 ~ $ ./server
FIFO Server...
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
server timeout, 30 - seconds passed
mvmalashenko@dk8n78 ~ $
```

рис.10 Канал

Сообщения передалась с периодичностью в несколько секунд, а по истечении 30 секунд - канал закрылся, и вывелось сообщение об этом.

Что будет в случае, если сервер завершит работу, не закрыв канал?

При завершении программы каналы автоматически закрываются.

# Вывод

Я приобрела практические навыки работы с именованными каналами.

---

## Контрольные вопросы

1. Именованные каналы отличаются от неименованных наличием идентификатора канала, который представлен как специальный файл (соответственно имя именованного канала — это имя файла).
2. Создание неименованного канала из командной строки возможно командой `pipe`.
3. Создание именованного канала из командной строки возможно с помощью `mkfifo`.
4. Функция языка C, создающая неименованный канал: `int read(int pipe_fd, void *area, int cnt); int write(int pipe_fd, void *area, int cnt);` Первый аргумент этих вызовов - дескриптор канала, второй - указатель на область памяти, с которой происходит обмен, третий - количество байт. Оба вызова возвращают число переданных байт (или -1 - при ошибке).
5. Функция языка C, создающая именованный канал: `int mkfifo (const char *pathname, mode_t mode);` Первый параметр — имя файла, идентифицирующего канал, второй параметр маска прав доступа к файлу. Вызов функции `mkfifo()` создаёт файл канала (с именем, заданным макросом `FIFO_NAME`): `mkfifo(FIFO_NAME, 0600);`
6. При чтении меньшего числа байтов, возвращается требуемое число байтов, остаток сохраняется для следующих чтений. При чтении большего числа байтов, возвращается доступное число байтов. 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=EP1PE`) (если процесс не установил обработки сигнала `SIGPIPE`, производится обработка по умолчанию -- процесс завершается).
7. Два и более процессов могут читать и записывать в канал.
8. Функция `write` записывает `length` байтов из буфера `buffer` в файл, определенный дескриптором файла `fd`. Эта операция чисто 'двоичная' и без буферизации. При единице возвращает действительное число байтов. Функция `write` возвращает число действительно записанных в файл байтов или -1 при ошибке, устанавливая при этом `errno`.
9. Строковая функция `strerror` - функция языков C/C++, транслирующая код ошибки, который обычно хранится в глобальной переменной `errno`, в сообщение об ошибке, понятном человеку.

---

## Библиография

- The Linux Programmer's Guide: Named Pipes: <https://tldp.org/LDP/lpg/node15.html>
- Linux Journal: Introduction to Named Pipes: <https://www.linuxjournal.com/article/2156>
- MSDN Library: Named Pipes: <https://docs.microsoft.com/ru-ru/windows/win32/ipc/named-pipes?redirectedfrom=MSDN>
- Programing with named pipes (from Sun and for Solaris, but a general enough intro for anyone):  
[https://web.archive.org/web/20120626103458/http://developers.sun.com/solaris/articles/named\\_p](https://web.archive.org/web/20120626103458/http://developers.sun.com/solaris/articles/named_p)