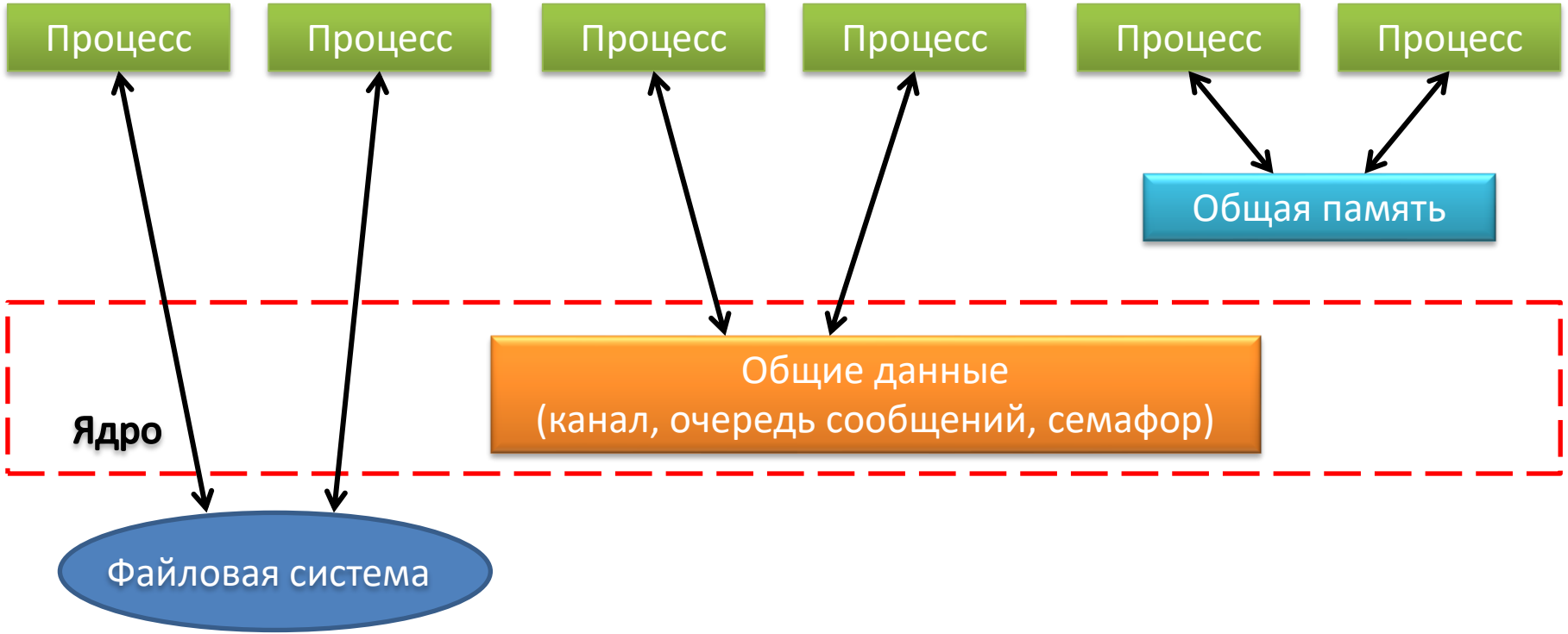
# Взаимодействие процессов

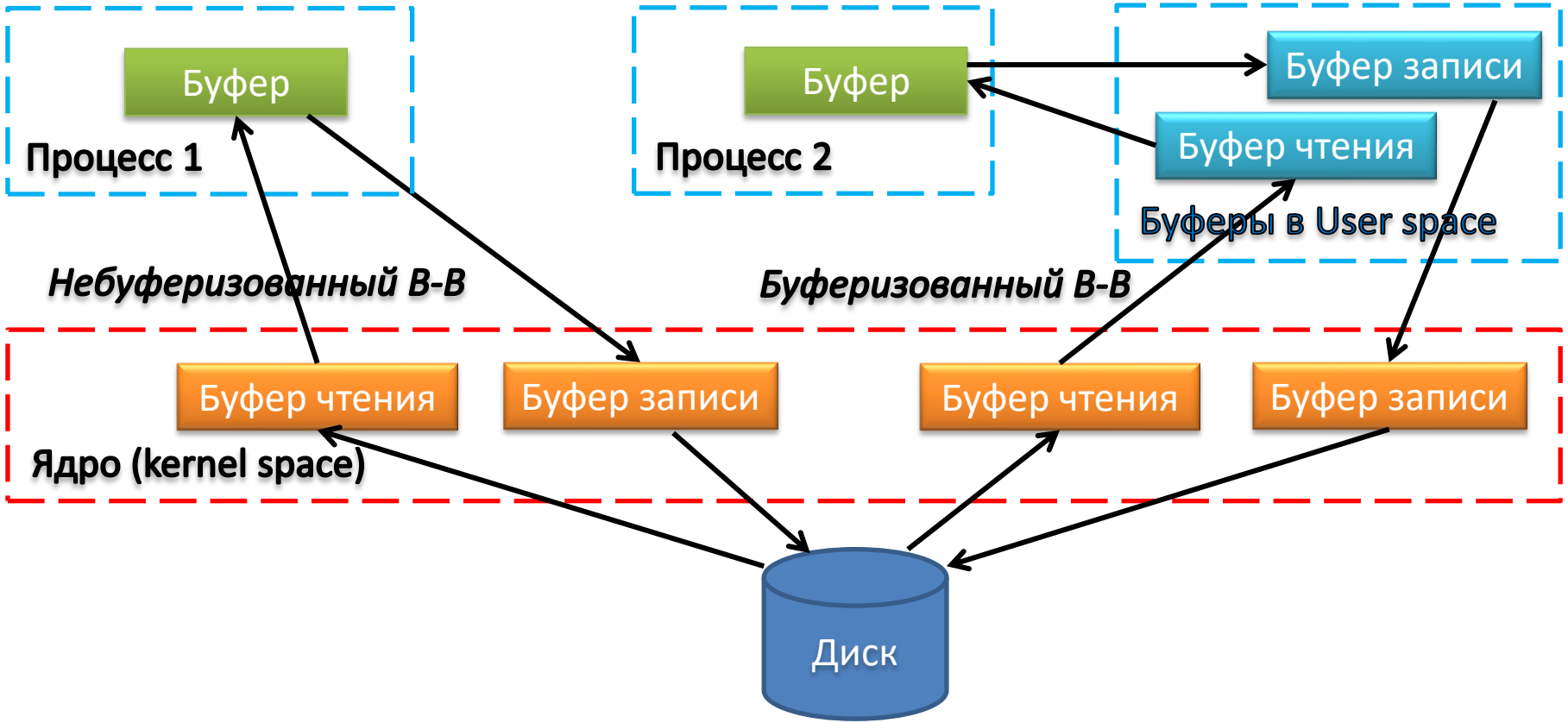
- Работа с файлами
- Каналы
- Сигналы
- Разделяемая память
- System V и POSIX
- Удаленный вызов процедур
- Сетевое взаимодействие

# Совместное использование данных



# Файлы

# Файловый ввод-вывод



# Файловый ввод-вывод

Небуферизованный (man 2)

- open(…)
- close(…)
- read(…)
- write(…)
- lseek(…)

Буферизованный (man 3)

- fopen(…)
- fdopen(…) / fileno(…)
- fclose(…)
- fread(…)
- fwrite(…)
- fflush(…)
- ~~fpurge(…)~~
- fseek(…)
- ftell(…)
- rewind (…)
- setvbuf(…)
- posix_madvise(…)

# Стандартные потоки

- Fd extern FILE ***stdin**; // 0
- Fd extern FILE ***stdout**; // 1
- Fd extern FILE ***stderr**; // 2, не буферизуется

- **Задание 2.1** (1 балл). Написать 2 программы: первая дописывает в файл строку, введенную пользователем. Вторая выводит содержимое файла. Имя файла указывается в параметрах запуска приложения. Использовать небуферизованный ввод-вывод.

- **Задание 2.2** (1 балл). Изменить программы из задания 2.1 так, чтобы использовался буферизованный ввод-вывод.

# Работа с каталогами

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

- DIR ***opendir**(const char *name);
  Открывает поток каталога с именем name и возвращает указатель на этот поток.

- int **closedir**(DIR *dir);
  Закрывает поток, связанный с каталогом dir.

- struct dirent ***readdir**(DIR *dir);
  Возвращает указатель на следующую запись каталога в структуре dirent, прочитанную из потока каталога dir. Возвращает NULL по достижении последней записи или если была обнаружена ошибка.

# Работа с каталогами

```
struct dirent {
    long d_ino;
    off_t d_off;
    unsigned short d_reclen;
    char d_name [NAME_MAX+1];
}
```

d_ino - номер inode
d_off - смещение записи dirent от начала каталога
d_reclen - длина d_name без учета последнего нуля
d_name - имя файла (строка, оканчивающаяся нулем).

# Работа с каталогами

- int **dirfd**(DIR *dir);
  Возвращает файловый дескриптор потока каталога dir.

- void **rewinddir**(DIR *dir);
  Устанавливает новую позицию потока каталога dir в начале каталога.

- void **seekdir**(DIR *dir, off_t offset);
  Устанавливает позицию следующего вызова readdir() в потоке каталога.

- off_t **telldir**(DIR *dir);
  Возвращает текущее положение в потоке каталогов dir.

# Работа с каталогами

- **Задание 2.3** (1 балл). Написать программу, выводящую содержимое указанного каталога.

- **Задание 2.4** (1 балл). Изменить программы из задания 2.1 или 2.2. При начале работы программа должна проверить, существует ли файл в текущем каталоге. Если файл отсутствует, выдается предупреждение и предлагается создать файл.

# Статус файла

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

- int **stat**(const char *file_name, struct stat *buf);
  Возвращает информацию о файле file_name и заполняет буфер buf.

- int **fstat**(int filedes, struct stat *buf);
  Возвращается информация об открытом файле, на который указывает filedes.

- int **lstat**(const char *file_name, struct stat *buf);
  В случае символьных ссылок возвращает информацию о самой ссылке, а не о файле, на который она указывает.

# Статус файла

```
struct stat {
  dev_t      st_dev;       /* устройство */
  ino_t      st_ino;       /* inode */
  mode_t     st_mode;      /* режим доступа */
  nlink_t    st_nlink      /* количество жестких ссылок */
  uid_t      st_uid;       /* идентификатор пользователя-владельца */
  gid_t      st_gid;       /* идентификатор группы-владельца */
  dev_t      st_rdev;      /* тип устройства  (если это устройство) */
  off_t      st_size;      /* общий размер в байтах */
  blksize_t  st_blksize;  /* размер блока ввода-вывода в файловой системе */
  blkcnt_t   st_blocks;   /* количество выделенных блоков */
  time_t     st_atime;    /* время последнего доступа */
  time_t     st_mtime;  /* время последней модификации */
  time_t     st_ctime;    /* время последнего изменения */
};
```

# Статус файла

Макросы POSIX для проверки файла:

- S_ISLNK(m) – символьная ссылка (нет в POSIX.1-1996.)
- S_ISREG(m) – обычный файл
- S_ISDIR(m) – каталог
- S_ISCHR(m) – символьное устройство
- S_ISBLK(m) – блочное устройство
- S_ISFIFO(m) – канал FIFO
- S_ISSOCK(m) – сокет

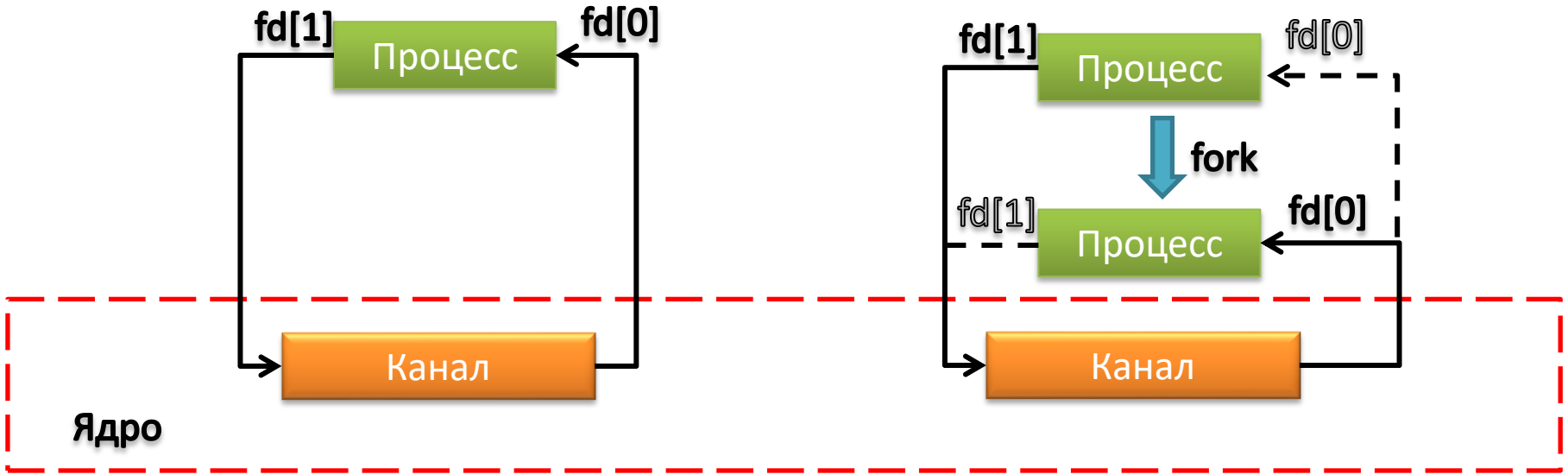
# Статус файла

- **Задание 2.5** (1 балл). Написать программу (усовершенствовать решение 2.3), выводящую содержимое указанного каталога с указанием типа файлов.

- **Задание 2.6** (2 балла). Написать простой файловый менеджер, позволяющий просматривать содержимое каталогов и отображающий типы файлов.

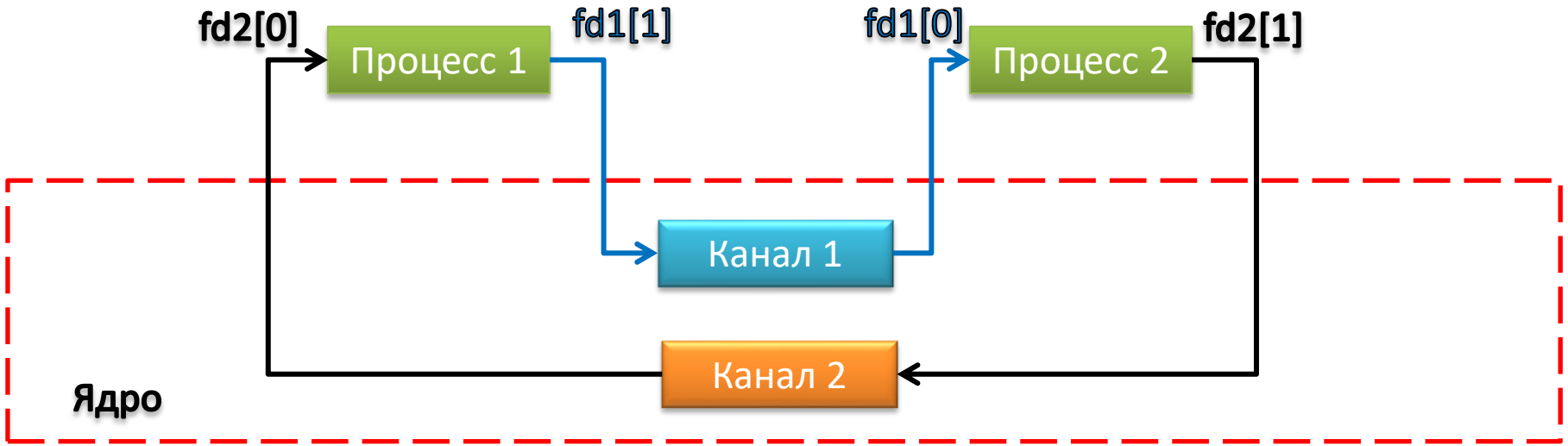
# Каналы

# Неименованные каналы



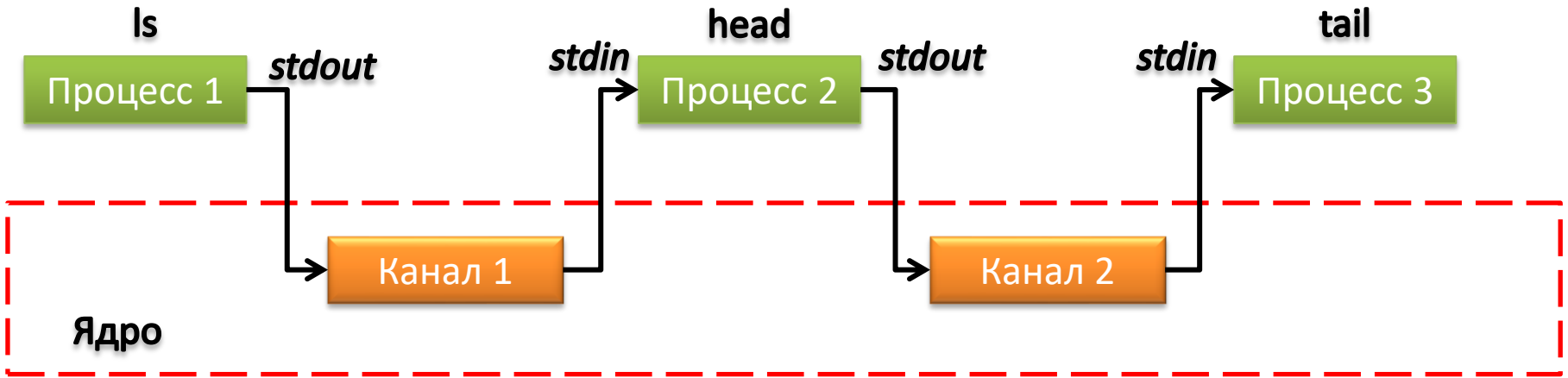

**Канал – это буфер FIFO**

# Двунаправленный обмен



# Программные каналы

**ls | head -3 | tail -1**



# Каналы

- Создание **неименованного** канала
  ```
  int pipe(int pipefd[2]);
  ```
  pipefd[0] – read, pipefd[1] - write

- Создание **именованного** канала
  ```
  int mkfifo(const char *pathname, mode_t mode);
  int mknod(char *pathname, int mode, int dev);
  ```

  Если при открытии FIFO через open() не указать режим O_NONBLOCK, то открытие FIFO блокируется и для записи, и для чтения. При записи канал блокируется до тех пор, пока другой процесс не откроет FIFO для чтения. При чтении канал снова блокируется до тех пор, пока другой процесс не запишет данные.

- Копирование файлового дескриптора
  ```
  int dup(int oldfd);
  int dup2(int oldfd, int newfd);
  ```

# Задания

**2.7** (2 балла). Написать программу, порождающую дочерний процесс и использующую однонаправленный обмен данными. Процесс-потомок генерирует случайные числа и отправляет родителю. Родитель выводит числа на экран и в файл. Количество чисел задается в параметрах запуска приложения.

**2.8** (2 балла). Скорректировать решение задачи 2.7 так, чтобы использовался двунаправленный обмен: процесс-родитель отправляет дочернему процессу ответное сообщение. В ответе - число, умноженное на 2.

**2.9** (3 балла). Усовершенствовать программу 1.5: добавить перенаправление ввода-вывода (возможность записи команд вида ls | head -3 | tail -1).

# Именованные каналы

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main()
{
  int fd_fifo; /*дескриптор FIFO*/
  char buffer[]="Текстовая строка для fifo\n";
  char buf[100];

  /*Если файл существует, удалим его*/
  unlink("/tmp/fifo0001.1");

  /*Создаем FIFO*/
  if((mkfifo("/tmp/fifo0001.1", O_RDWR)) == -1)
  {
    fprintf(stderr, "Невозможно создать fifo\n");
    exit(0);
  }

  /*Открываем fifo для чтения и записи*/
  if((fd_fifo=open("/tmp/fifo0001.1", O_RDWR)) == -
1)
  {
    fprintf(stderr, "Невозможно открыть fifo\n");
    exit(0);
  }
```

# Именованные каналы

```
  write(fd_fifo, buffer, strlen(buffer));

  if(read(fd_fifo, &buf, sizeof(buf)) == -1)
    fprintf(stderr, "Невозможно прочесть из FIFO\n");
  else
    printf("Прочитано из FIFO: %s\n",buf);
  return 0;
}
```

**Задание 2.10** (2 балла). Сделать две программы: первая передает в именованный канал случайные числа, а вторая – считывает данные и выводит на экран. Что будет если запустить несколько экземпляров каждой программы?