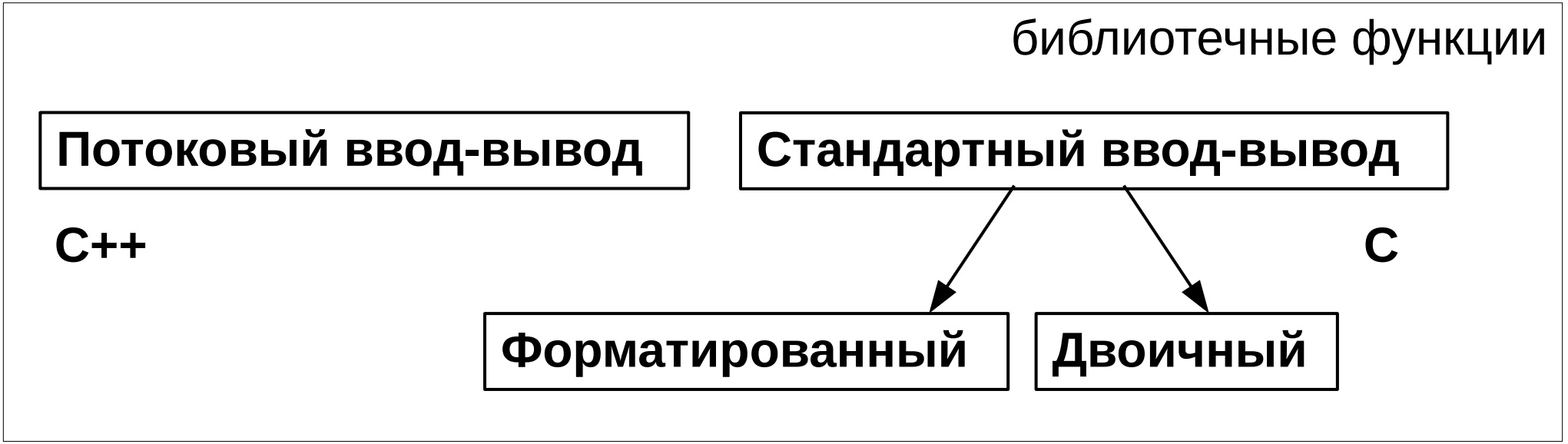
Ввод-вывод системного уровня

Средства ввода-вывода

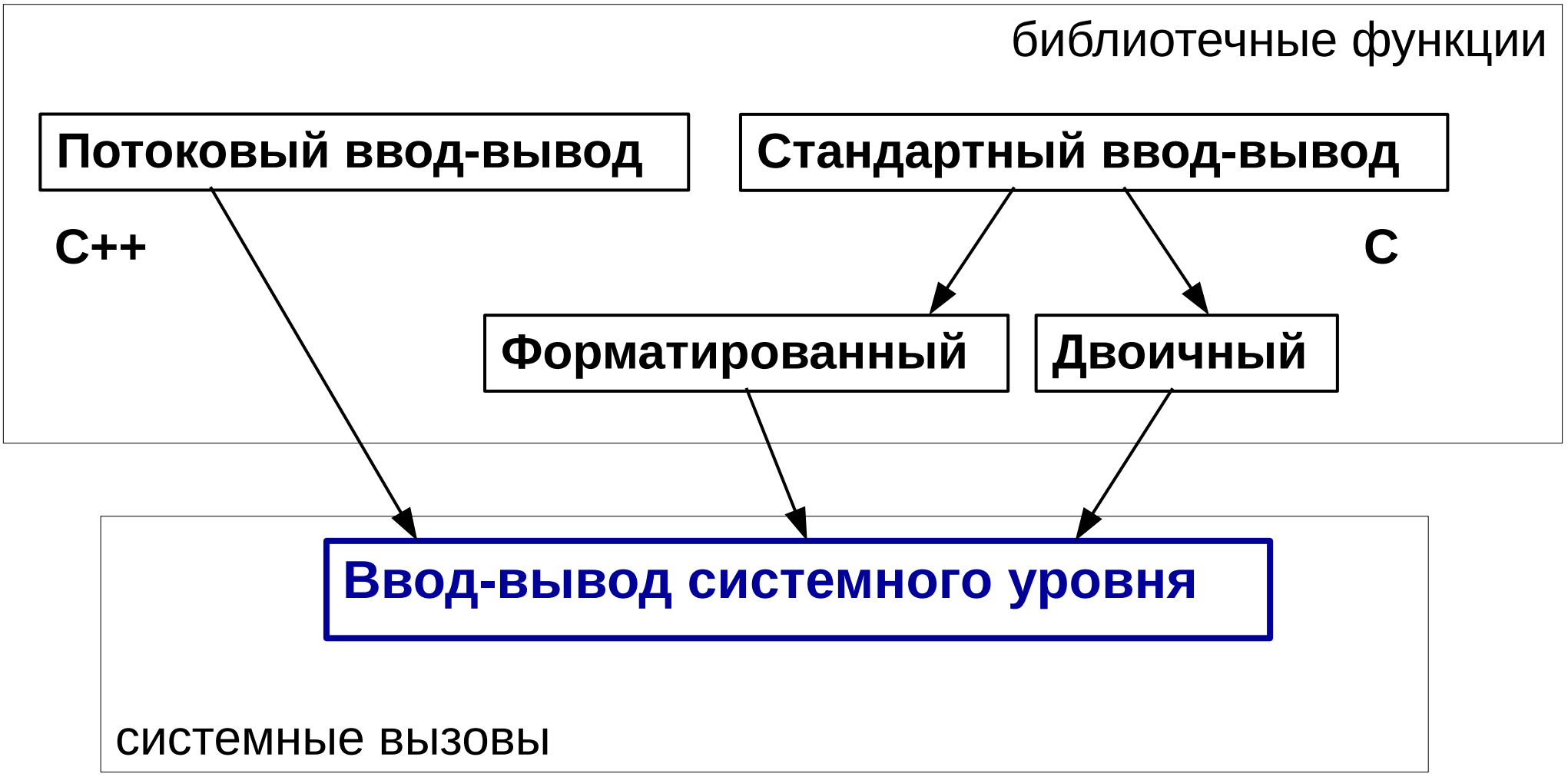
библиотечные функции

Потоковый ввод-вывод	Стандартный ввод-вывод
C++	C

Средства ввода-вывода



Средства ввода-вывода



Сравнительный анализ

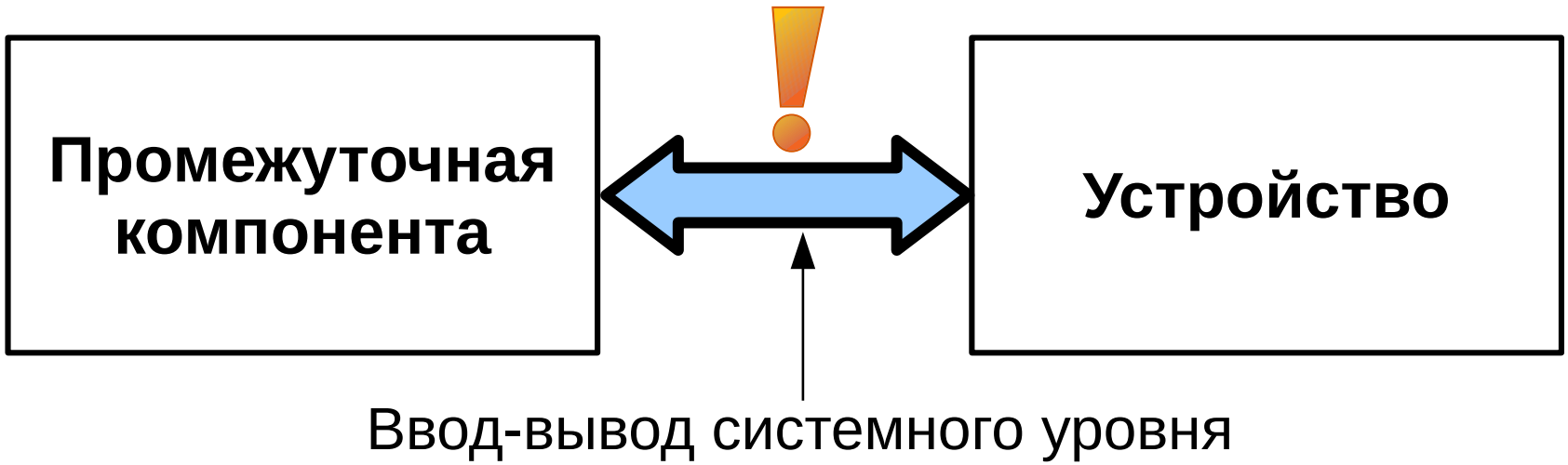
- ✔**Эффективность (без посредников)**
- ✔**Дополнительные возможности**
- ✔**Удобен для работы с устройствами**

Сравнительный анализ

- ✔ **Эффективность (без посредников)**
- ✔ **Дополнительные возможности**
- ✔ **Удобен для работы с устройствами**
- ✔ **Ориентирован на двоичные данные (без элементов форматирования)**
- ✔ **Уровень квалификации разработчика**

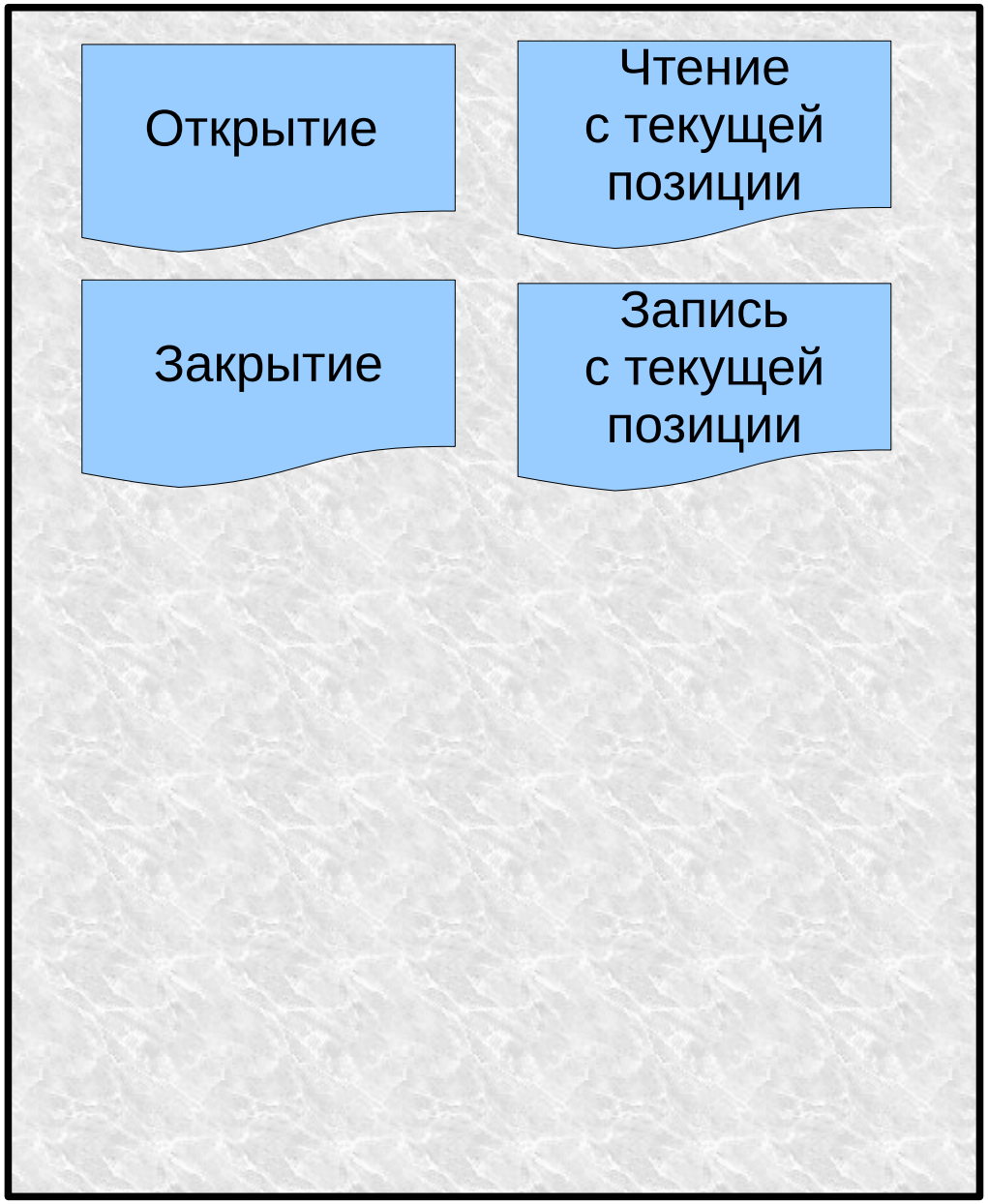
Сравнительный анализ

- ✔ **Эффективность (без посредников)**
- ✔ **Дополнительные возможности**
- ✔ **Удобен для работы с устройствами**
- ✔ **Ориентирован на двоичные данные (без элементов форматирования)**
- ✔ **Уровень квалификации разработчика**



Самое простое и важное

Операции над файлами — основа системного программирования



Самое простое и важное

Операции над файлами — основа системного программирования

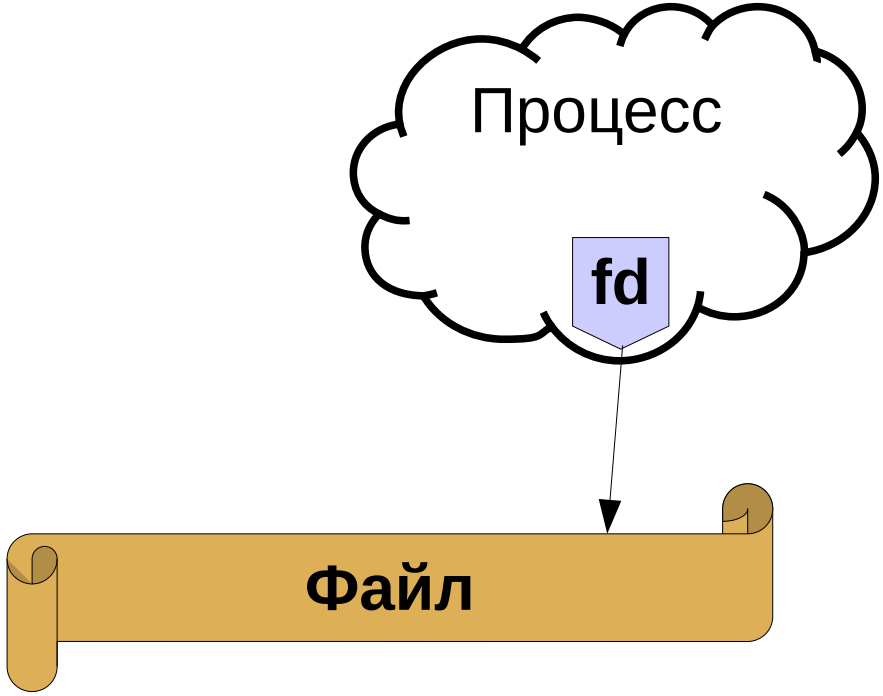
Самое простое и важное

Операции над файлами — основа системного программирования

Самое простое и важное

Операции над файлами — основа системного программирования

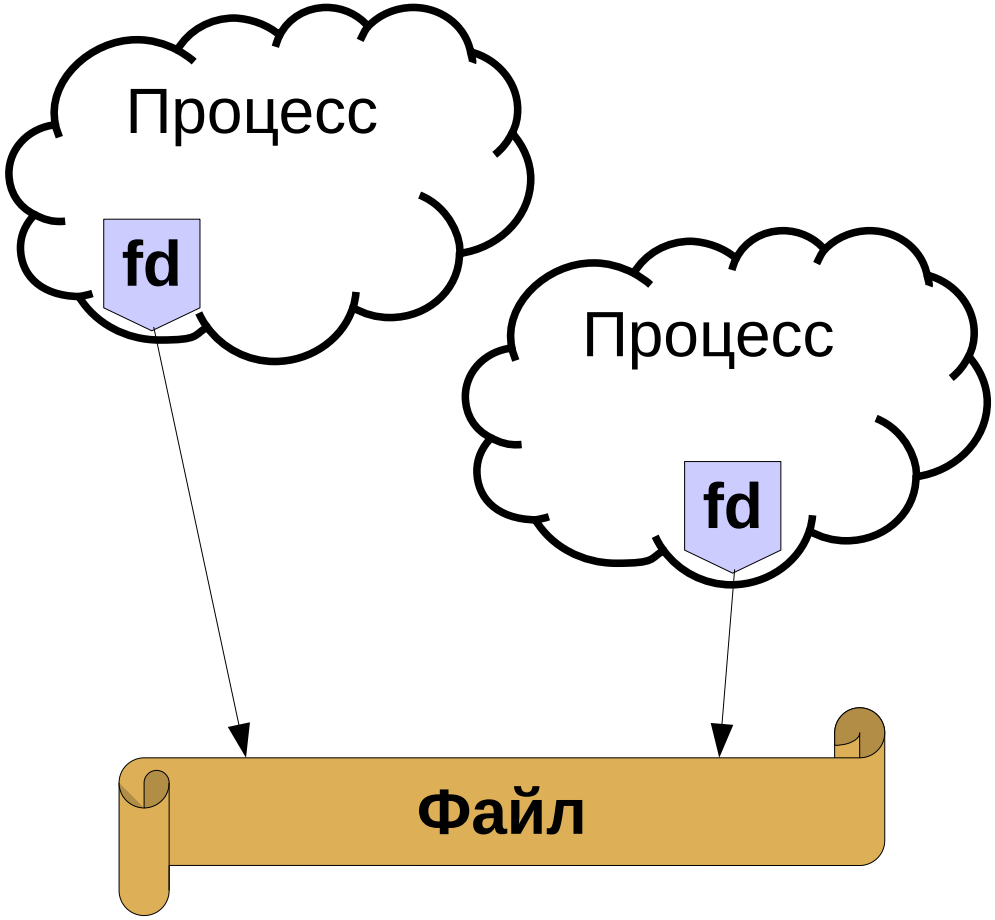
- Открытие
- Чтение с текущей позиции
- Закрытие
- Запись с текущей позиции
- Позиционирование
- Усечение
- Синхронизация
- Мультиплексирование
- Управление
- Ожидание события



Самое простое и важное

Операции над файлами — основа системного программирования

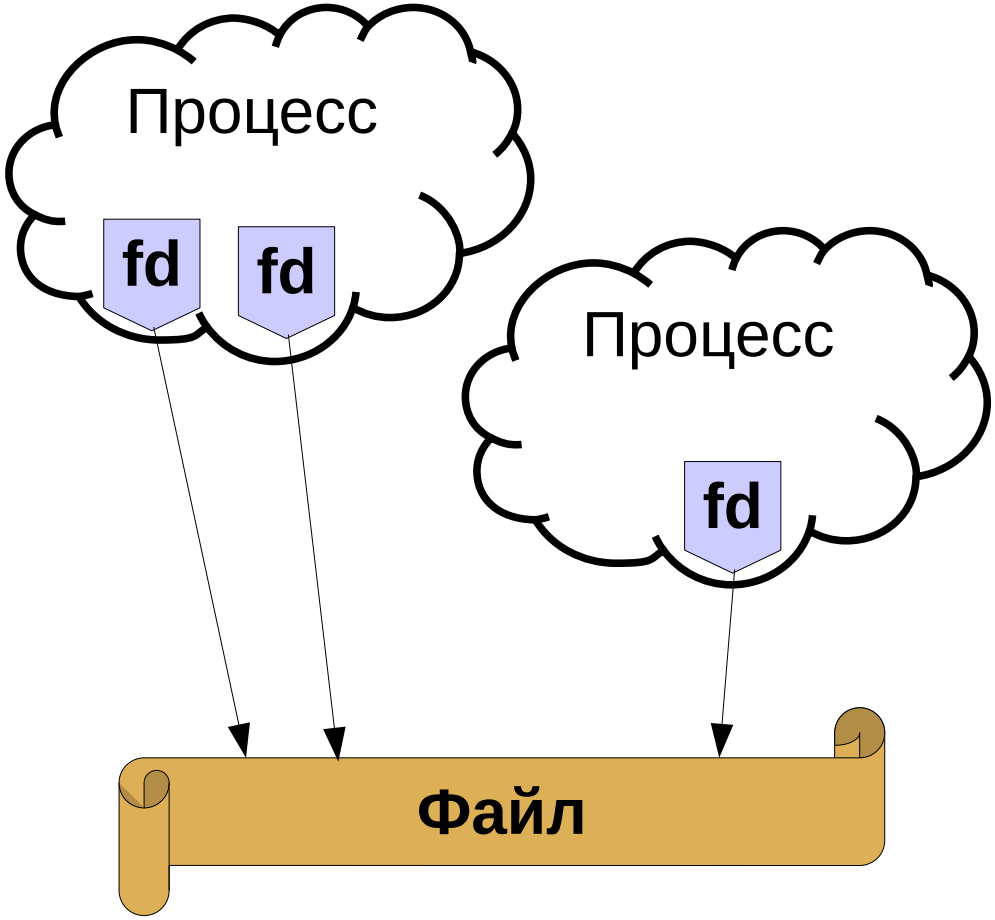
- Открытие
- Чтение с текущей позиции
- Закрытие
- Запись с текущей позиции
- Позиционирование
- Усечение
- Синхронизация
- Мультиплексирование
- Управление
- Ожидание события



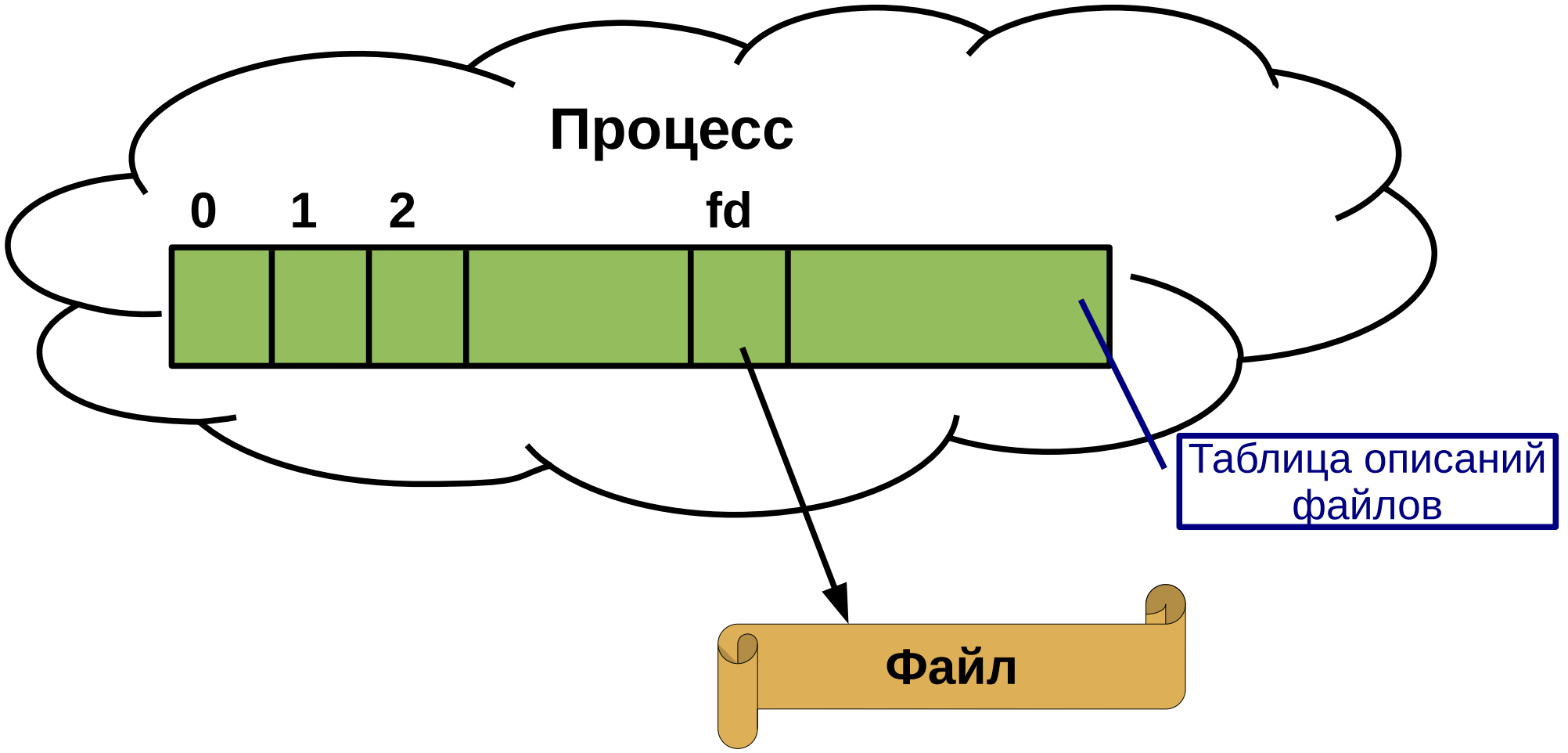
Самое простое и важное

Операции над файлами — основа системного программирования

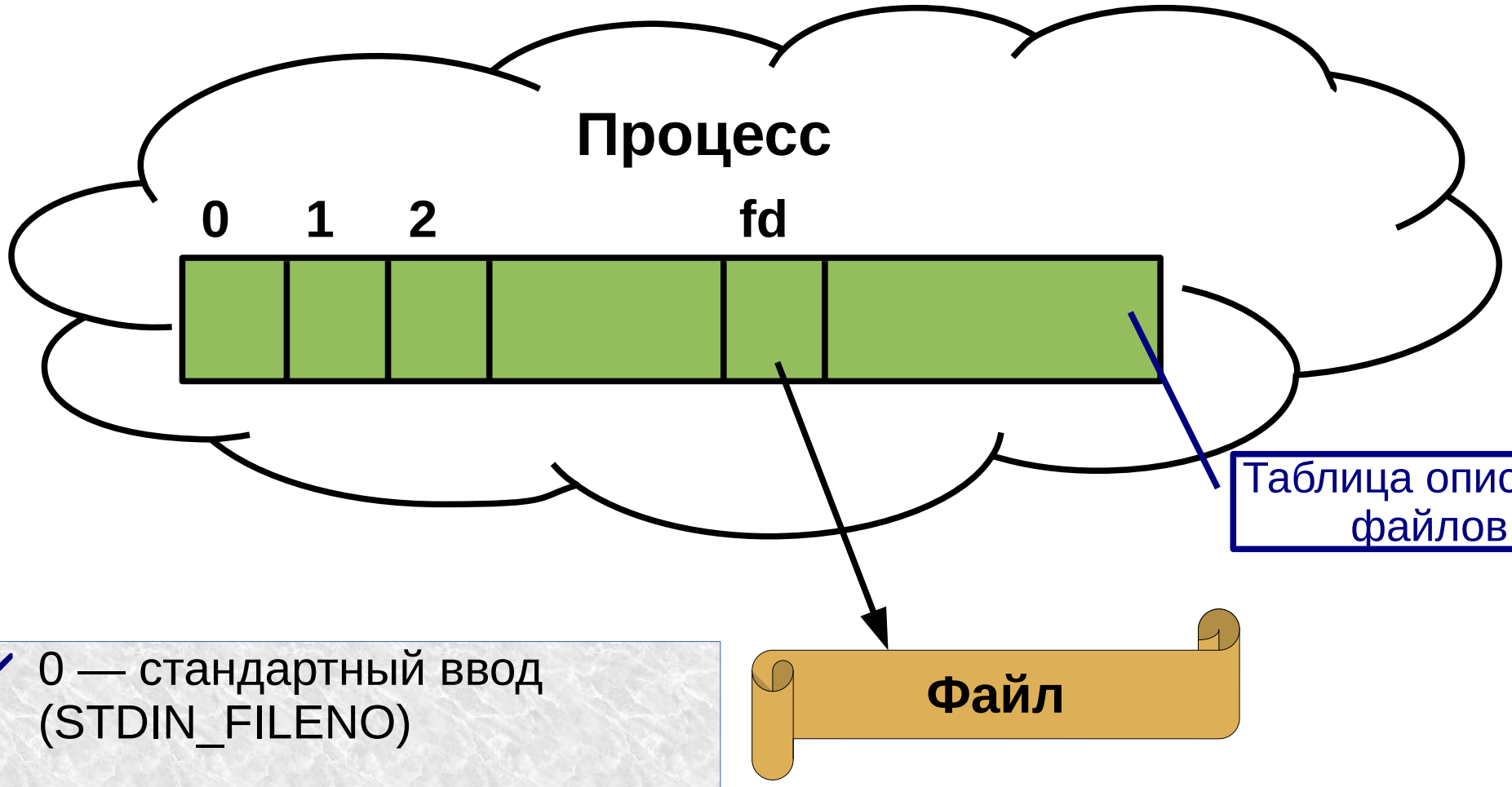
- Открытие
- Чтение с текущей позиции
- Закрытие
- Запись с текущей позиции
- Позиционирование
- Усечение
- Синхронизация
- Мультиплексирование
- Управление параметрами
- Ожидание события



Файловый дескриптор



Файловый дескриптор



- 0 — стандартный ввод (STDIN_FILENO)
- 1 — стандартный вывод (STDOUT_FILENO)
- 2 — вывод сообщений об ошибках (STDERR_FILENO)

Открытие файлов

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

int open(const char* name, int flags);
int creat(const char* name, mode_t mode);
```

- **name** — путь к файлу
- **flags** — управление режимом открытия
- **mode** — задание атрибутов доступа при создании
- Возвращают дескриптор

```
O_RDONLY  O_WRONLY  O_RDWR
O_APPEND  O_TRUNC   O_CREAT  O_NONBLOCK
```

Примеры использования

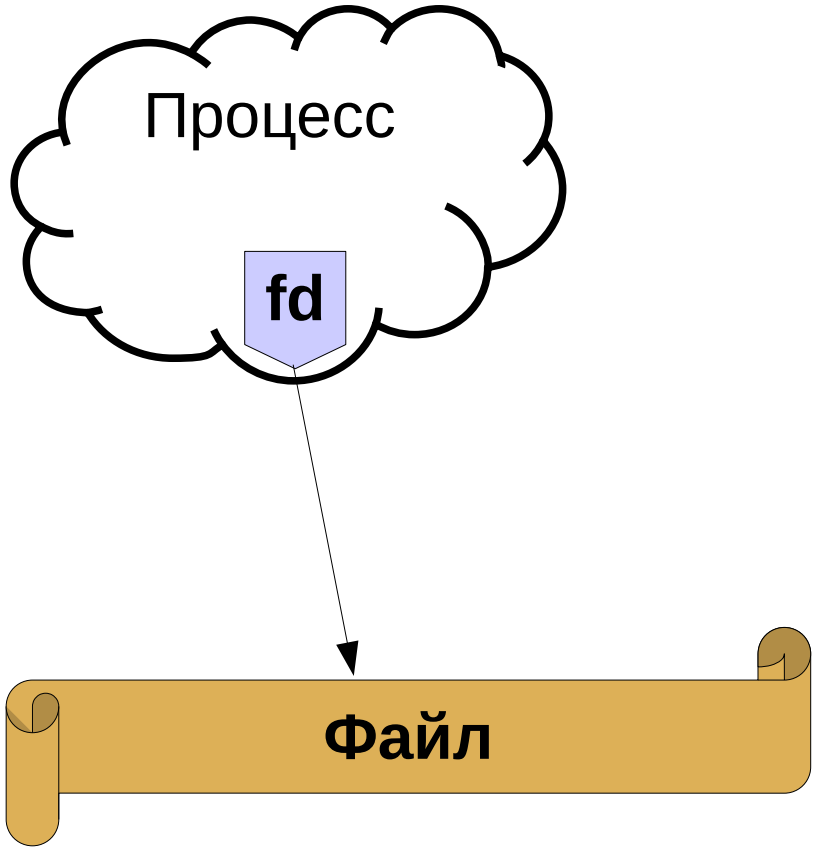
```
int fd;
fd = open(«/home/user/mylog»,O_WRONLY|O_APPEND);
if ( fd == -1 ) {
  /*обработка ошибки*/
}
```

Примеры использования

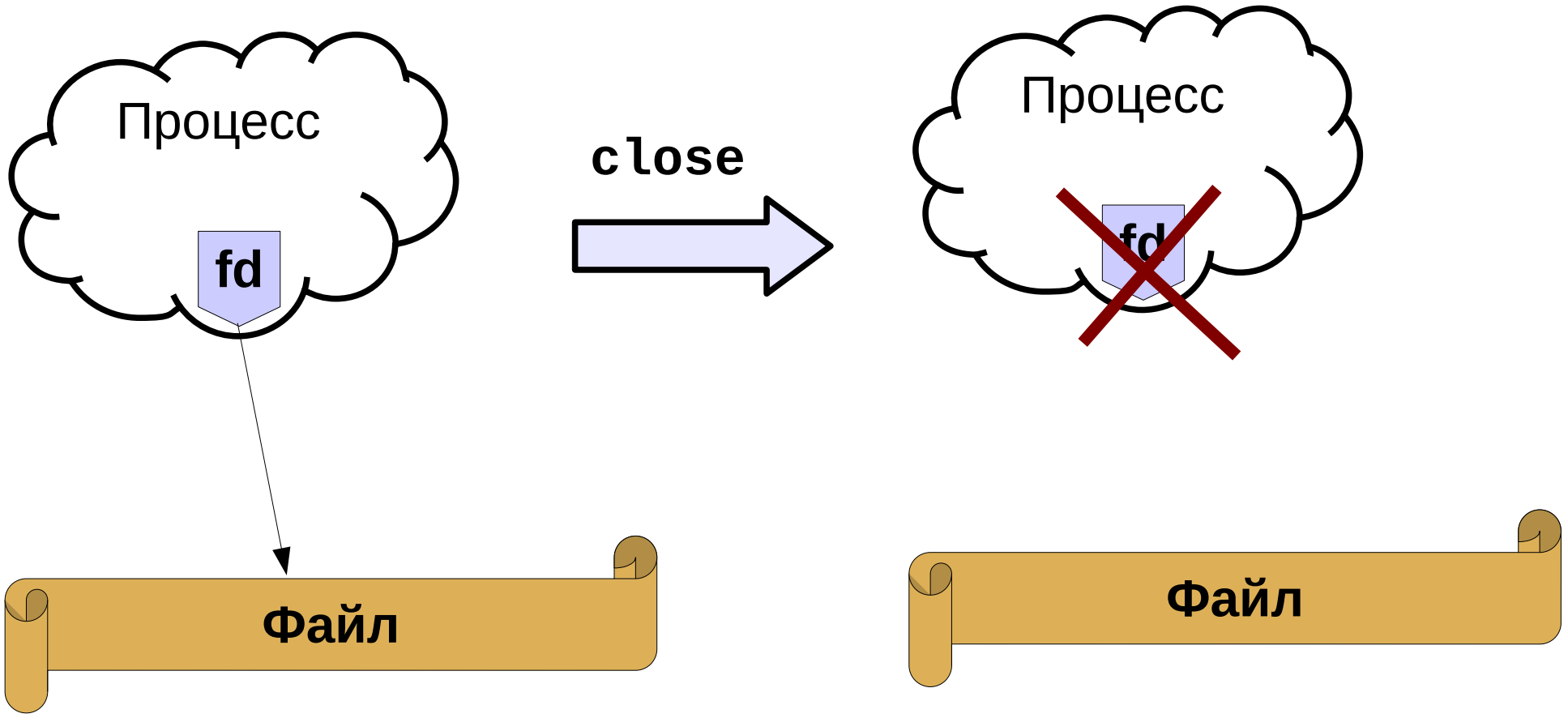
```
int fd;
fd = open(«/home/user/mylog»,O_WRONLY|O_APPEND);
if ( fd == -1 ) {
  /*обработка ошибки*/
}
```

```
int fd;
fd = creat(«/home/user/results.dat»,0644);
if ( fd == -1 ) {
  /*обработка ошибки*/
}
```

Закрытие файла



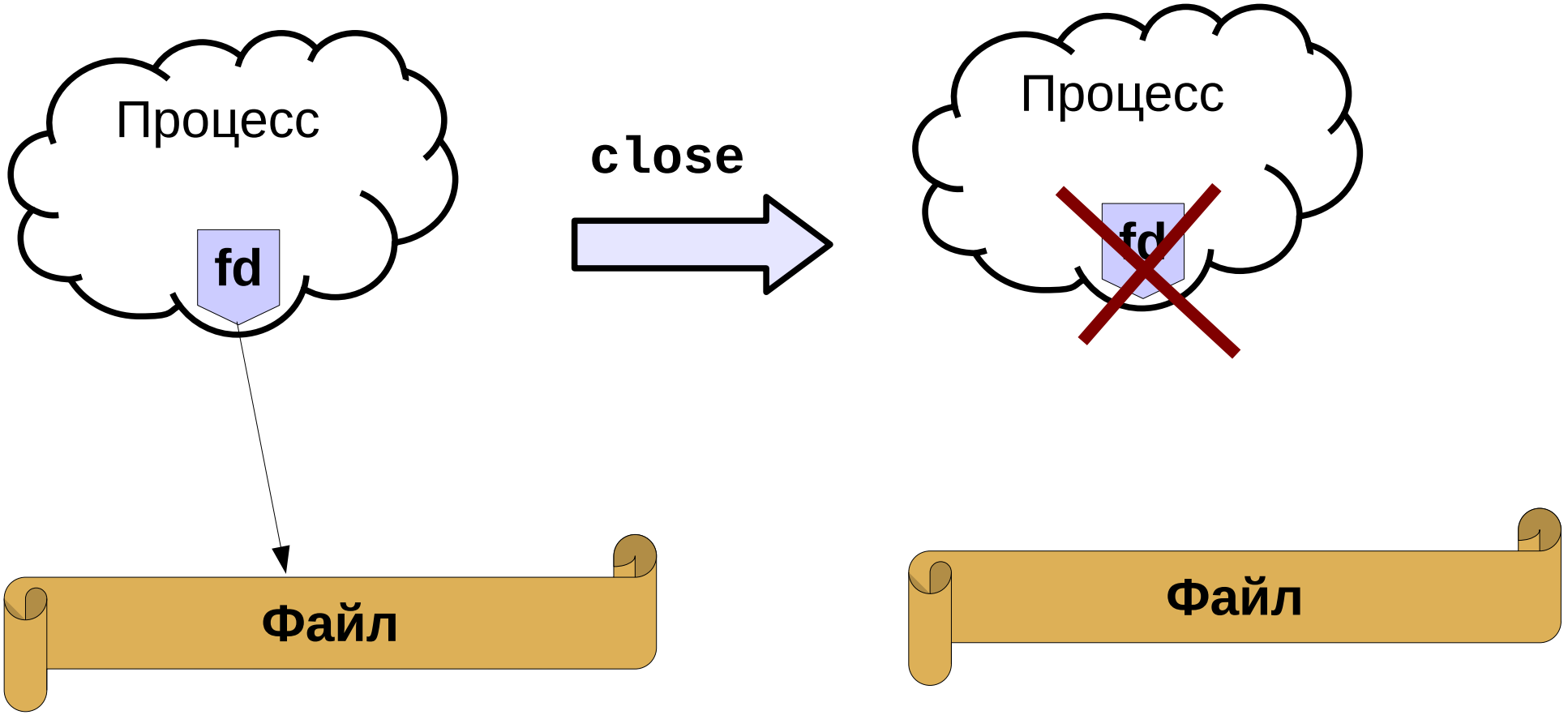
Закрытие файла



Закрытие файла

```
#include <unistd.h>

int close(int fd);
```

- ✔ **fd** — дескриптор
- ✔ Возвращает 0 при успехе или -1 при ошибке



Управление параметрами

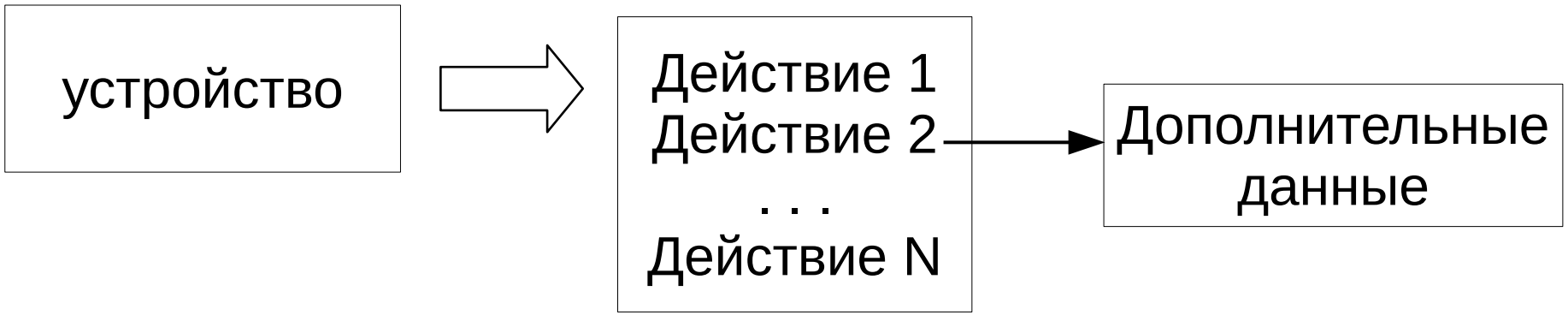
```
#include <sys/ioctl.h>

int ioctl(int fd, int request, ...);
```

- **`fd`** — файловый дескрпитор
- **`request`** — запрашиваемое действие
- **`...`** — список дополнительных данных, переменного размера
- Возвращает 0 при успехе, -1 при ошибке

Управление параметрами

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

int ioctl(int fd, int request, ...);
```

- ✔ **`fd`** — файловый дескрпитор
- ✔ **`request`** — запрашиваемое действие
- ✔ **`...`** — список дополнительных данных, переменного размера
- ✔ Возвращает 0 при успехе, -1 при ошибке



Пример использования **ioctl** (1)


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

Пример использования **ioctl** (1)

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

int cdrom_eject(const char* device);
```

Пример использования **ioctl** (1)


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

int cdrom_eject(const char* device);

int main (int argc, char *argv[])
{
   int ret = cdrom_eject("/dev/cdrom");
   return ret;
}
```

Пример использования **ioctl** (2)

```
int cdrom_eject(const char* device) {

    int fd = open (device, O_RDONLY | O_NONBLOCK);
    if (fd < 0) {
        perror ("open");
        return -1;
    }
```

Пример использования **ioctl** (2)

```
int cdrom_eject(const char* device) {

    int fd = open (device, O_RDONLY | O_NONBLOCK);
    if (fd < 0) {
        perror ("open");
        return -1;
    }

    int ret = ioctl (fd, CDROMEJECT, 0);
    if (ret) {
        perror ("ioctl");
        return -2;
    }
```

Пример использования **ioctl** (2)

```
int cdrom_eject(const char* device) {

    int fd = open (device, O_RDONLY | O_NONBLOCK);
    if (fd < 0) {
        perror ("open");
        return -1;
    }

    int ret = ioctl (fd, CDROMEJECT, 0);
    if (ret) {
        perror ("ioctl");
        return -2;
    }

    close (fd);
    return 0;
}
```

Чтение из файла

```
#include <unistd.h>

ssize_t read(int fd, void* buf, size_t len);
```

- **fd** — дескриптор
- **buf** — область сохранения
- **len** — сколько байт прочитать
- Возвращает число считанных байт или -1 при ошибке

Чтение из файла

```
#include <unistd.h>

ssize_t read(int fd, void* buf, size_t len);
```

- **fd** — дескриптор
- **buf** — область сохранения
- **len** — сколько байт прочитать
- Возвращает число считанных байт или -1 при ошибке

```
unsigned long word;
ssize_t count;
count = read(fd,&word,sizeof(unsigned long));
if ( count == -1 )
  /*обработка ошибки*/
```

Проблемы при чтении

- ✔ **Прочитано меньше, чем ожидалось**
- ✔ **Повторение вызова**

Проблемы при чтении

- ✔ **Прочитано меньше, чем ожидалось**
- ✔ **Ничего не прочитано**
- ✔ **Повторение вызова**
- ✔ **Без рецепта**

Проблемы при чтении

- ✔ **Прочитано меньше, чем ожидалось**
- ✔ **Ничего не прочитано**
- ✔ **Не завершается операция**
- ✔ **Повторение вызова**
- ✔ **Без рецепта**
- ✔ **Дождаться поступления данных**

Проблемы при чтении

- ✔ **Прочитано меньше, чем ожидалось**
- ✔ **Повторение вызова**
- ✔ **Ничего не прочитано**
- ✔ **Без рецепта**
- ✔ **Не завершается операция**
- ✔ **Дождаться поступления данных**
- ✔ **Ошибка EAGAIN (EWOULDBLOCK)**
- ✔ **Повторение вызова позже**

Проблемы при чтении

✔ Прочитано меньше, чем ожидалось	✔ Повторение вызова
✔ Ничего не прочитано	✔ Без рецепта
✔ Не завершается операция	✔ Дождаться поступления данных
✔ Ошибка EAGAIN (EWOULDBLOCK)	✔ Повторение вызова позже
✔ Ошибка EINTR	✔ Повторение вызова

Проблемы при чтении

✔ Прочитано меньше, чем ожидалось	✔ Повторение вызова
✔ Ничего не прочитано	✔ Без рецепта
✔ Не завершается операция	✔ Дождаться поступления данных
✔ Ошибка EAGAIN (EWOULDBLOCK)	✔ Повторение вызова позже
✔ Ошибка EINTR	✔ Повторение вызова
✔ Иная ошибка	✔ Завершиться??

Считывание всех байт

```
while ( len != 0 ) {
  ret = read(fd, buf, len);
```

Считывание всех байт

```
while ( len != 0 ) {
  ret = read(fd, buf, len);
  if ( ret == 0 )
    break;
```

Считывание всех байт

```
while ( len != 0 ) {
  ret = read(fd, buf, len);
  if ( ret == 0 )
    break;
  if ( ret == -1 ) {
    if ( errno == EINTR )
      continue;
```

Считывание всех байт

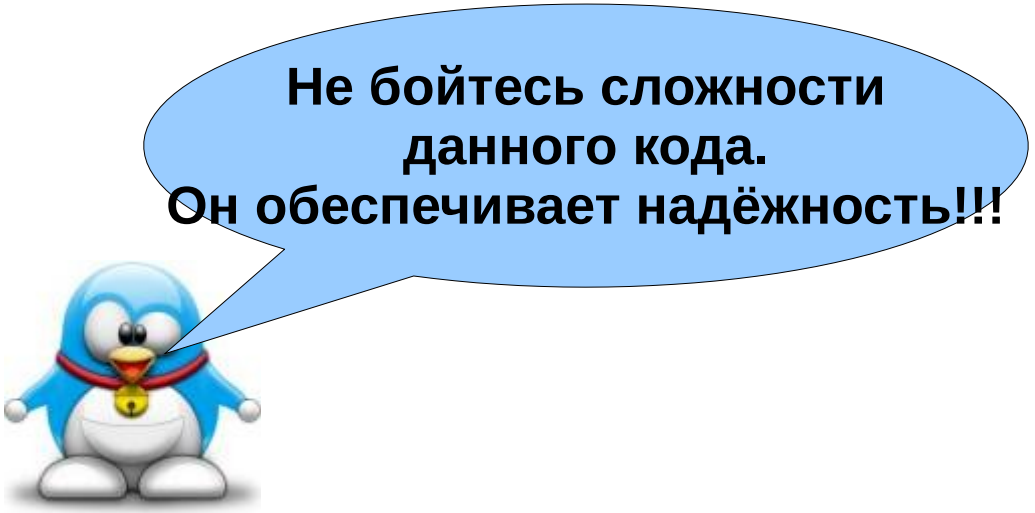
```
while ( len != 0 ) {
  ret = read(fd, buf, len);
  if ( ret == 0 )
    break;
  if ( ret == -1 ) {
    if ( errno == EINTR )
      continue;
    if ( errno == EAGAIN ||
         errno == EWOULDBLOCK  )
      break;
```

Считывание всех байт

```
while ( len != 0 ) {
  ret = read(fd, buf, len);
  if ( ret == 0 )
    break;
  if ( ret == -1 ) {
    if ( errno == EINTR )
      continue;
    if ( errno == EAGAIN ||
         errno == EWOULDBLOCK  )
      break;

    perror(«read»);
    break;
  }
```

Считывание всех байт

```
while ( len != 0 ) {
  ret = read(fd, buf, len);
  if ( ret == 0 )
    break;
  if ( ret == -1 ) {
    if ( errno == EINTR )
      continue;
    if ( errno == EAGAIN ||
         errno == EWOULDBLOCK  )
      break;

    perror(«read»);
    break;
  }

  len -= ret;
  buf += ret;
}
```

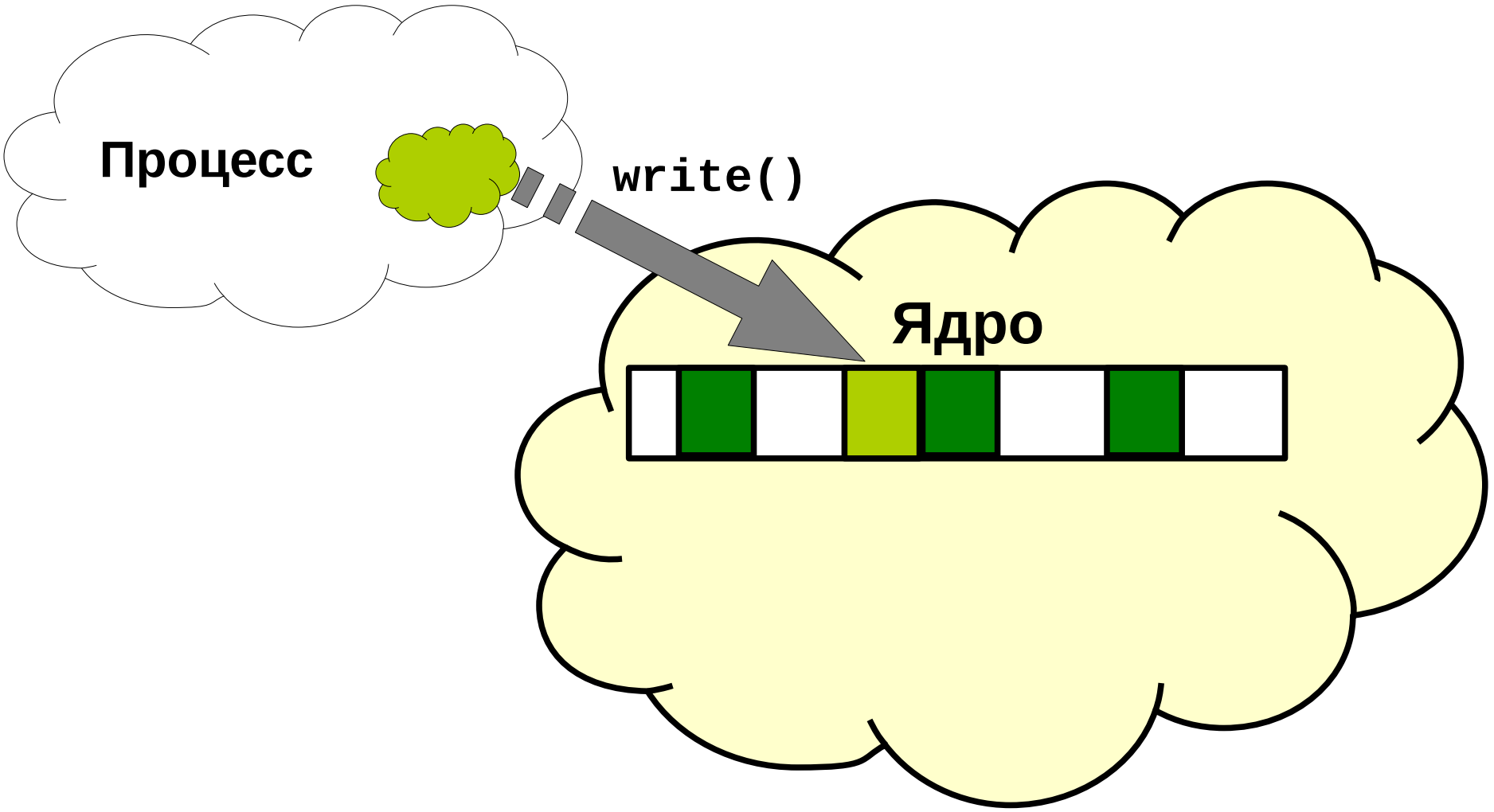


Запись в файл

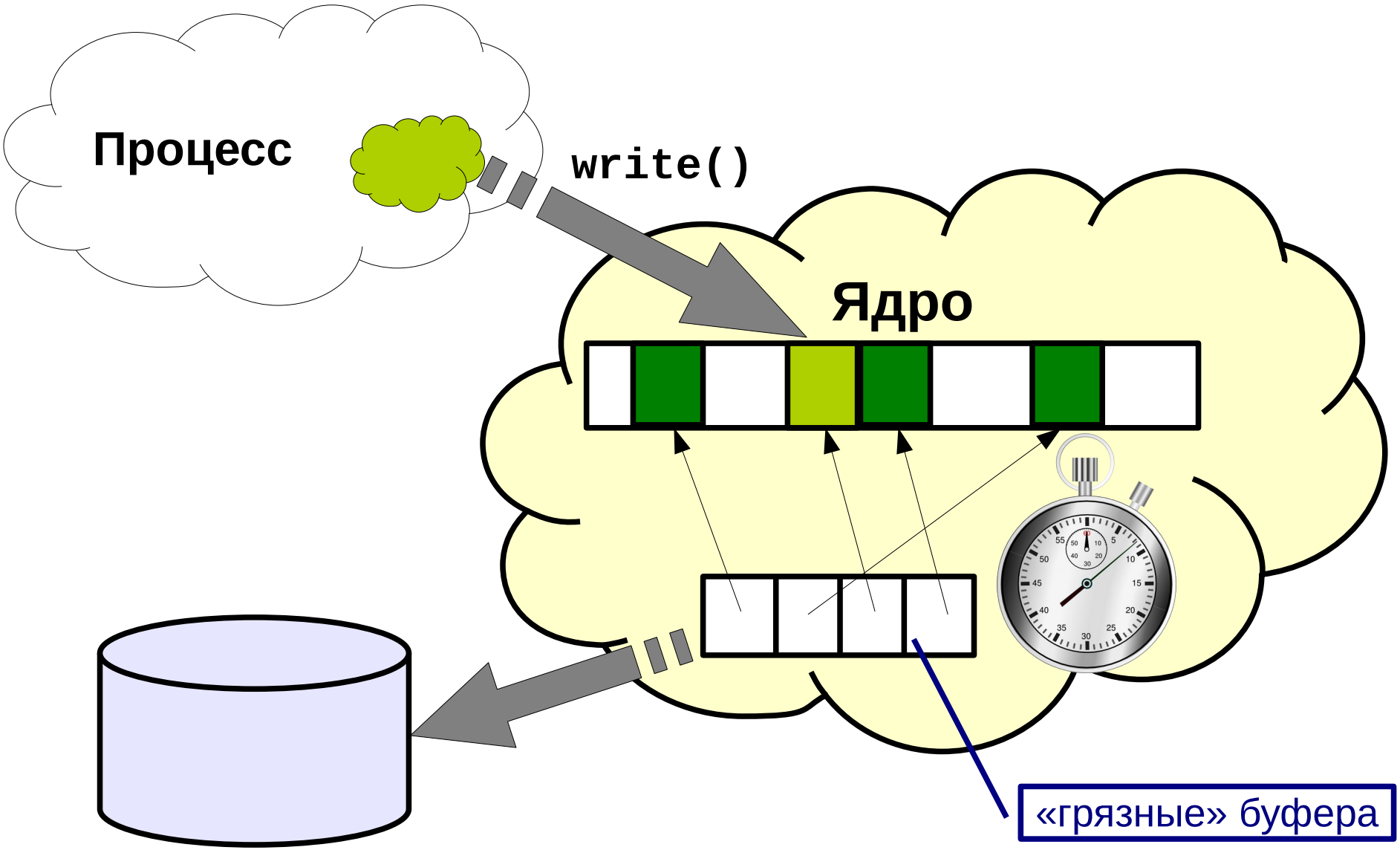
```
#include <unistd.h>

ssize_t write(int fd, const void* buf, size_t len);
```

- ✔ **fd** — дескриптор
- ✔ **buf** — область данных
- ✔ **len** — сколько байт записать
- ✔ Возвращает число записанных байт или -1 при ошибке

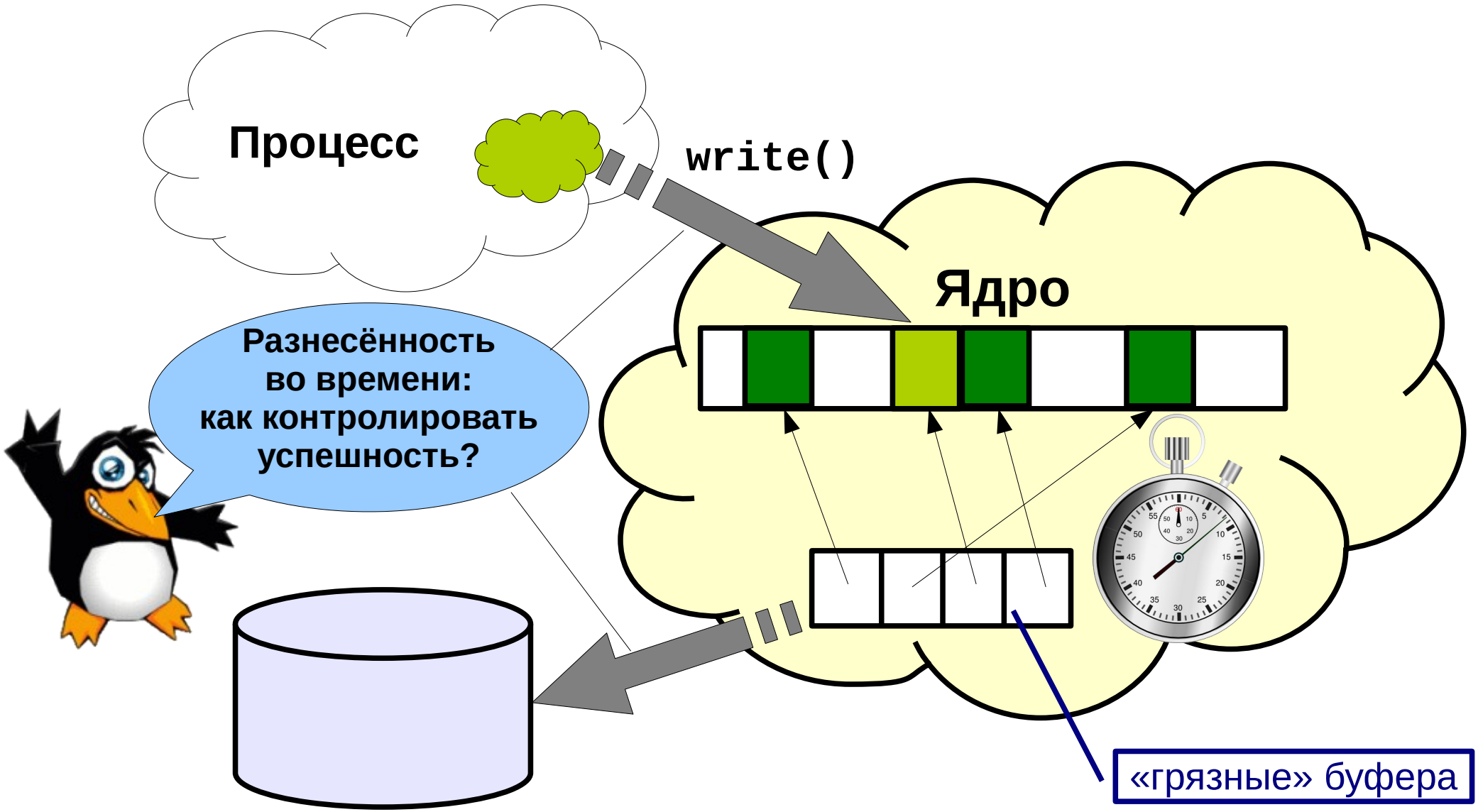
Отложенная запись



Отложенная запись



Отложенная запись



Перемещение по файлу

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

off_t lseek(int fd, off_t pos, int origin);
```

- **fd** — дескриптор
- **pos** — смещение в байтах относительно базы
- **origin** — базовое положение
- Возвращает новое положение от начала

Перемещение по файлу

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

off_t lseek(int fd, off_t pos, int origin);
```

- ✔ **fd** — дескриптор
- ✔ **pos** — смещение в байтах относительно базы
- ✔ **origin** — базовое положение
- ✔ Возвращает новое положение от начала

SEEK_CUR SEEK_SET SEEK_END

Перемещение по файлу

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

off_t lseek(int fd, off_t pos, int origin);
```

- ✔ **fd** — дескриптор
- ✔ **pos** — смещение в байтах относительно базы
- ✔ **origin** — базовое положение
- ✔ Возвращает новое положение от начала

SEEK_CUR SEEK_SET SEEK_END

```
lseek(fd, (off_t)0, SEEK_SET);
lseek(fd, (off_t)0, SEEK_END);
off_t pos = lseek(fd, (off_t)0, SEEK_CUR);
```

Ввод-вывод системного уровня