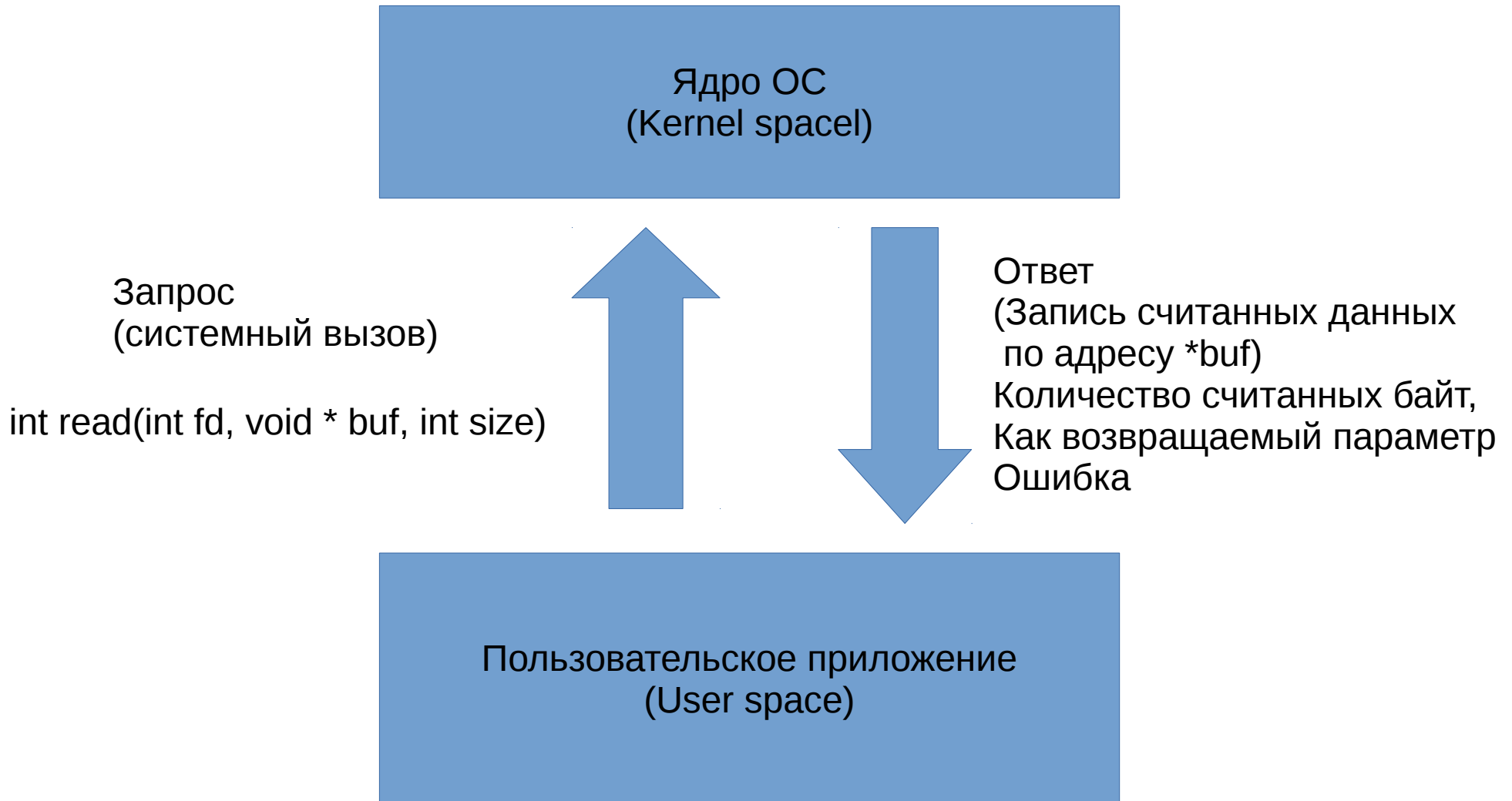
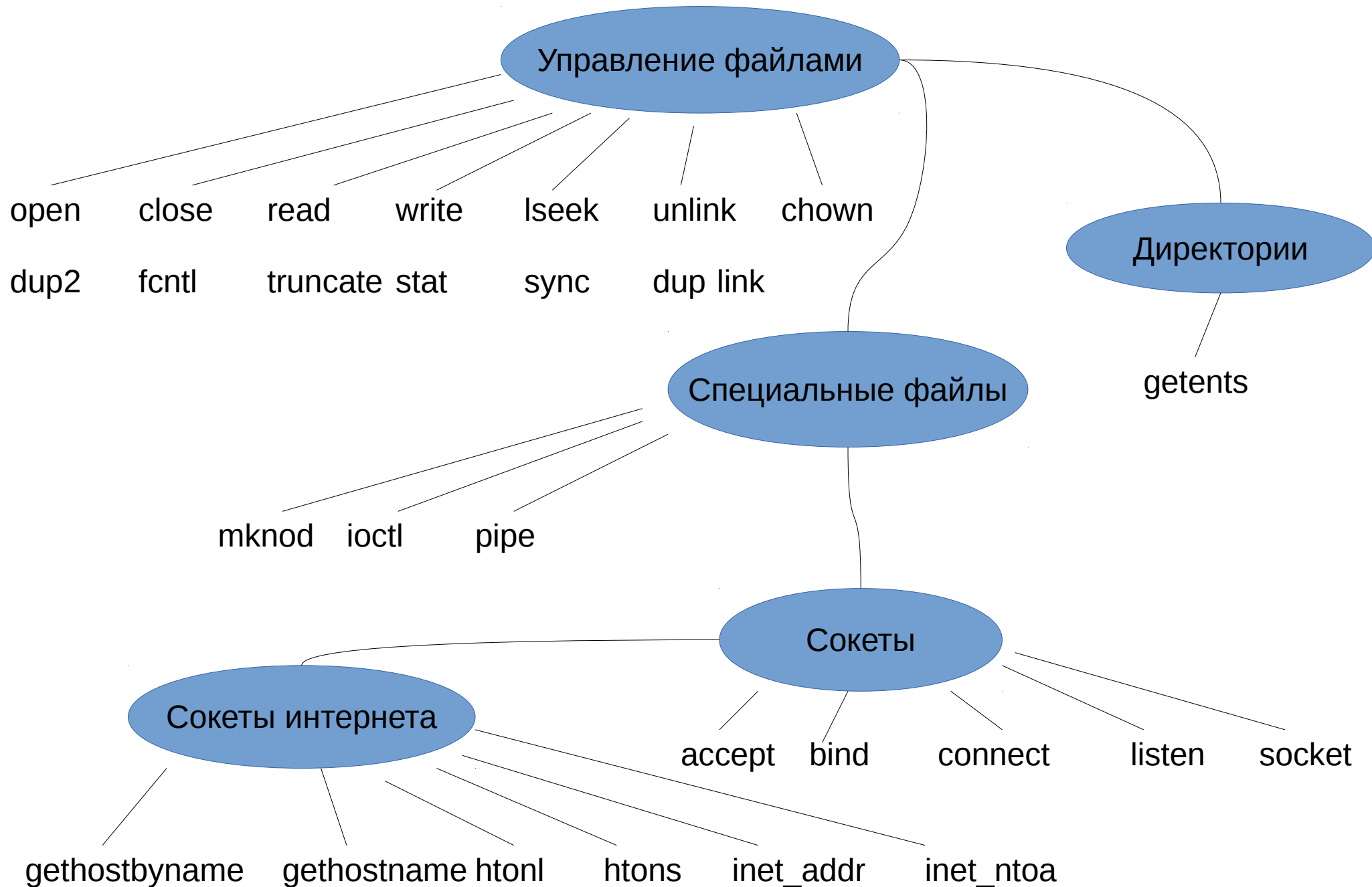


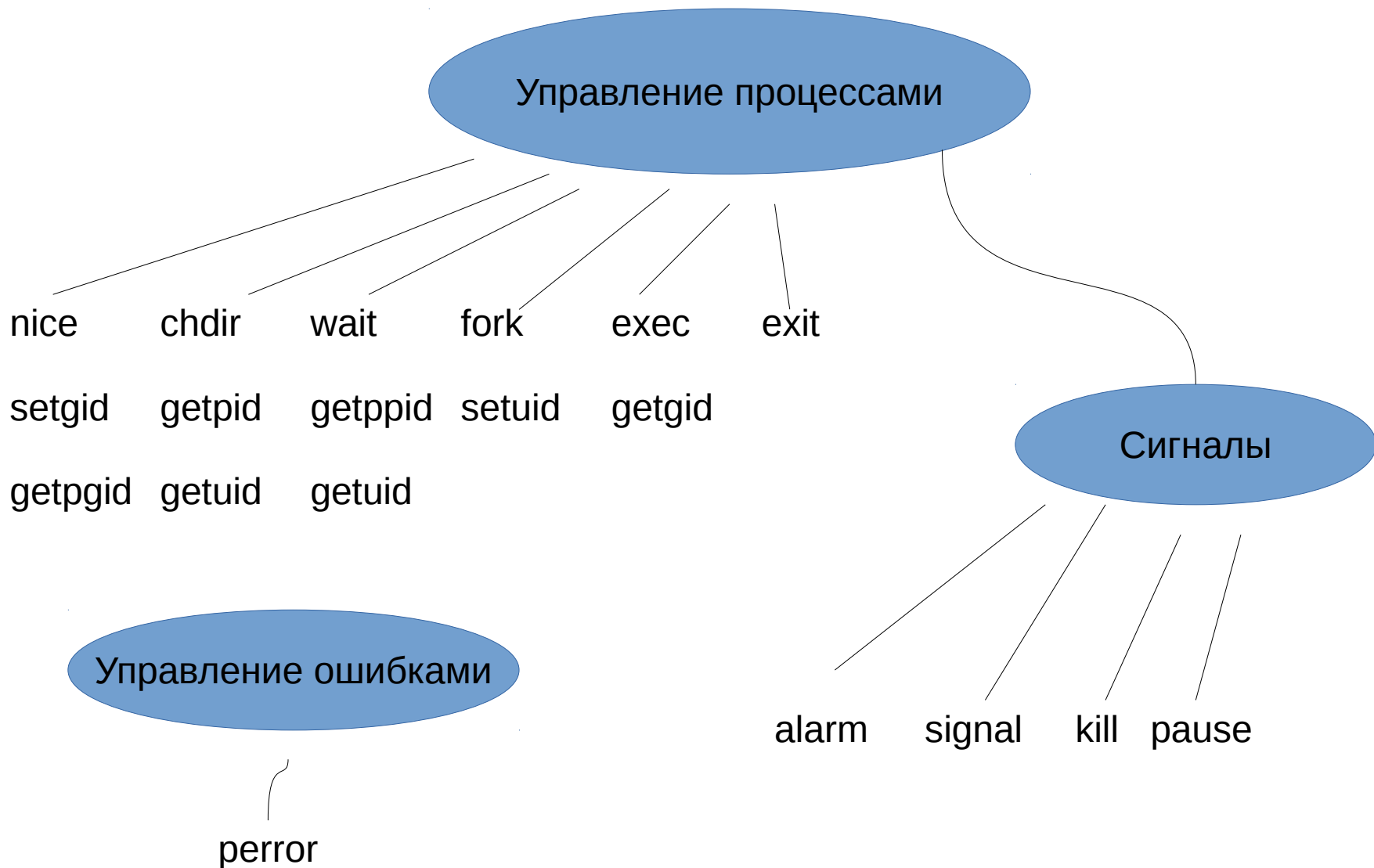
Системные вызовы



Иерархия системных вызовов



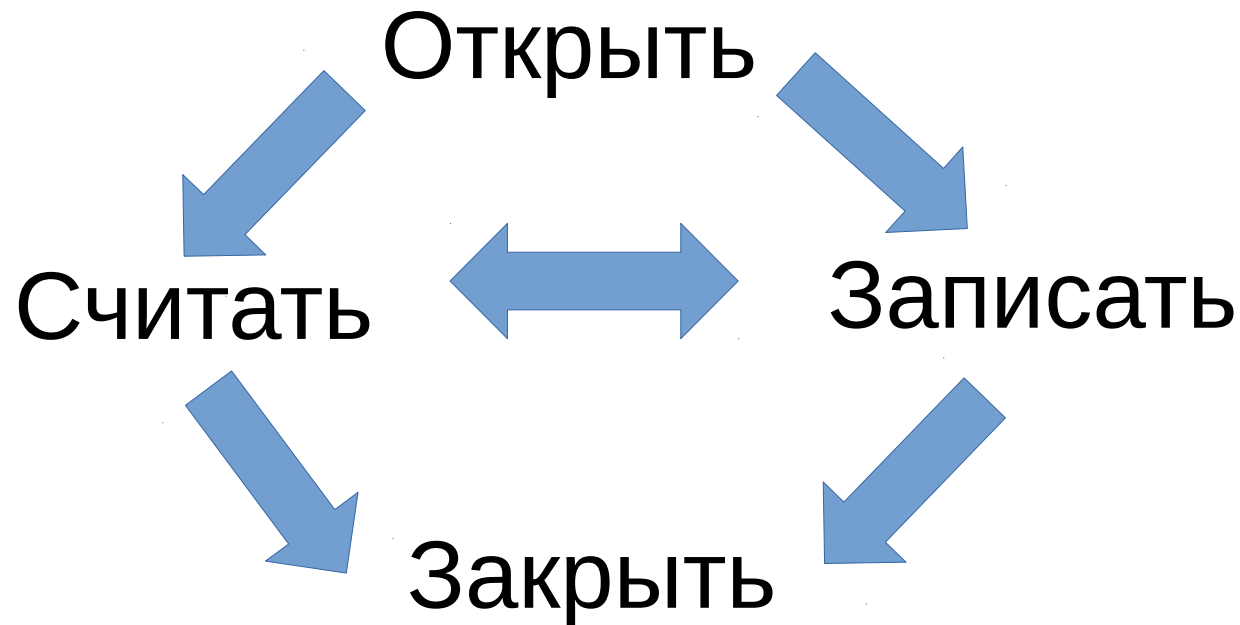
Иерархия системных вызовов (продолжение)



Файлы и файловая система

Основной принцип файловой модели:
«Всё есть файл»

Общий интерфейс при работе с любыми файлами



Типы файлов

- Обычные файлы
- Специализированные файлы
 - Файлы драйверов
 - Сокеты
 - Именованные каналы
- Каталоги и ссылки
 - Жесткие ссылки
 - Символьные ссылки

Открытие файла

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

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

Определение `mode_t`

Определения режимов `mode`

Определения флагов `flags`

Имя файла (**name**): "hello.txt", "/home/user/hello.txt"

Справка:

\$ man 2 open

Флаги (**flags**):

O_RDONLY — только чтение;

O_WRONLY — только запись;

O_RDWR — чтение/запись

Разрешения (**mode**):

S_IRWXU — пользователю чтение, запись и исполнение;

S_IRWXG — группе чтение, запись исполнение

S_IROTH — остальным только чтение

Открытие файла

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



Возвращает:

:) Значение файлового дескриптора в случае успешного выполнения

:(-1 в случае ошибки.

Файловый дескриптор

- Нужен ядру для поиска информации об открытом файле в собственной *файловой таблице*
- Файловые дескрипторы нумеруются с нуля, значения 0, 1, 2 зарезервированы:
 - 0 — стандартный ввод **stdin**;
 - 1 — стандартный вывод **stdout**;
 - 2 — стандартный поток ошибок **stderr**;
- Несколько файловых дескрипторов могут ссылаться на один файл.

Заккрытие файла

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

```
int close(int fd);
```

fd - Файловый дескриптор открытого файла

Возвращаемое значение

- «-1» в случае ошибки.
- «0» в случае успешного завершения

Справка:

```
$ man 2 close
```


Обработка ошибок

Согласно стандарту POSIX все системные функции в случае неуспешного выполнения возвращают значение «-1»

Код ошибки хранится в глобальной переменной **errno**, доступной через заголовочный файл **errno.h**

Получить текстовое сообщение об ошибке можно, вызвав функцию **perror()**

```
#include <errno.h>
```

```
int main()  
{
```

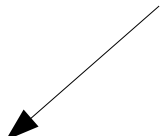
```
    if(somecall() == -1)  
    {
```

```
        int get_err = errno;  
        perror("somecall() error");
```


```
    }
```

```
}
```

Сохранение кода ошибки последнего вызова
(следующий может переписать значение)



Печать текста ошибки.
Аргумент функции — текст,
предшествующий тексту ошибки



Подробности (коды ошибок, особенности использования)

```
$ man errno
```

```
$ man perror
```

Чтение из файла

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

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

fd - Файловый дескриптор открытого файла

buf - Указатель на буфер куда ядру записать считанные символы

len - Количество считываемых байт

Возвращаемое значение:

Подробности:

В случае успеха:

\$ man 2 read

- Количество считанных байт
 - Равно значению **len** (Всё ОК)
 - Больше 0, но меньше **len**. (Процесс был прерван, дочитать можно след. Вызовом
- Значение «0» - конец файла. Считывать больше нечего

В случае ошибки:

Значение «-1»

- Значение `errno == EINTR`. Вызов был прерван до считывания байт, необходимо повторить
- Значение `errno == EAGAIN`. В настоящий момент нет доступных данных, вызов необходимо повторить позднее.

Пример: Побайтовое чтение из файла

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>

int main()
{
    int fd = open("hello.txt", O_RDONLY);
    if(fd > 0)
    {
        char symb;
        int res;
        do
        {
            res = read(fd, &symb, (sizeof(symb)));
            if(res > 0)
                printf("%c", symb);
        }
        while(res <= 0);
        /* Здесь необходимо обрабатывать ошибки чтения */
        close(fd);
    }
    else
    {
        perror("Error");
    }
    return 0;
}
```

Запись в файл

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

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

fd - Файловый дескриптор открытого файла

buf - Указатель на *константный* буфер откуда ядру брать данные для записи

len - Количество записываемых байт

Возвращаемое значение аналогично **read()**

- «-1» в случае ошибки
- Количество записанных байт в случае успешного завершения

Подробности:

```
$ man 2 write
```

Пример: Запись строки в файл

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>

#include <string.h> //определение strlen()

int main()
{
    int fd = open("hello.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if(fd > 0)
    {
        char out[100] = "Hello, world!\n";
        int out_len = strlen(out); //Определение размера строки
        int wrote_bytes = write(fd, out, out_len);
        if(wrote_bytes == out_len)
            printf("Success!\n");
        else
            printf("Error : (\n");
        close(fd);
    }
    else
    {
        perror("Error");
    }

    return 0;
}
```