# Каналы

# Каналы

Обеспечивают передачу информации в виде потока байтов без сохранения границ сообщений.

Каналы бывают:

- неименованные (каналы)
- именованные (FIFO-файлы)

# Неименованные каналы

Нет имени.

Взаимодействовать при помощи неименованных каналов могут только родственные (имеющие общего родителя или родитель и дочерний процесс) процессы.

# Системные вызовы

int pipe(int fd[2]);

int pipe2(int fd[2], int flags);

O_NONBLOCK (O_NDELAY)
O_CLOEXEC

# Последовательность действий

- Создать канал
- Создать процесс
- В каждом из процессов закрыть неиспользуемый дескриптор
- По окончании работы закрыть используемый дескриптор

Каналы предназначены только для передачи информации в одном направлении.

Каждый процесс должен иметь для каждого канала только один открытый дескриптор.

Для двустороннего обмена информацией следует использовать два канала.

# Правила для каналов (1)

- При чтении меньшего числа байтов, чем находится в канале, возвращается требуемое число байтов, остаток сохраняется для последующих чтений.

# Правила для каналов (2)

- При чтении большего числа байтов, чем находится в канале, возвращается доступное число байтов. Необходимо правильно обрабатывать данную ситуацию.

# Правила для каналов (3)

- Если канал пуст и ни один процесс не открыл его на запись, при чтении будет прочитано ноль байтов. Если хотя бы один процесс имеет дескриптор канала, открытый на запись, то читающий процесс будет заблокирован.

# Правила для каналов (4)

- Операция записи числа байтов меньшего емкости канала является атомарной. Данные от нескольких процессов, пишущих в канал, не перемешиваются.

# Правила для каналов (5)

- Операция записи числа байтов большего емкости канала блокирует процесс до освобождения места в канале.
- Запись в канал, не открытый ни одним процессом на чтение, приводит к посылке процессу сигнала SIGPIPE.

# Режим без блокировки

При создании канала или при помощи системного вызова fcntl файловый дескриптор может быть переведен в неблокирующий режим. В этом случае соответствующие системные вызовы возвращают специальную ошибку EAGAIN.

# Именованные каналы (1)

Сначала необходимо создать файл типа FIFO. Команда shell mkfifo или системный вызов mknod. При первом открытии этого файла создается канал. Другие процессы, открывая файл, присоединяются к существующему каналу.

# Именованные каналы (2)

При открытии именованного канала в блокирующем режиме процесс может быть переведен в состояние ожидания до появления другого процесса, открывающего тот же самый канал для противоположной операции.

# Именованные каналы (3)

В режиме без блокировки одностороннее открытие именованного канала возможно только на чтение. При попытке открытия именованного канала на запись при отсутствующем дескрипторе на чтение приведет к ошибке ENXIO.

# Именованные каналы (4)

После получения файлового дескриптора работа с именованным каналом ничем не отличается от неименованного. Канал уничтожается после того, как последний процесс закроет файловый дескриптор.

# Именованные каналы (5)

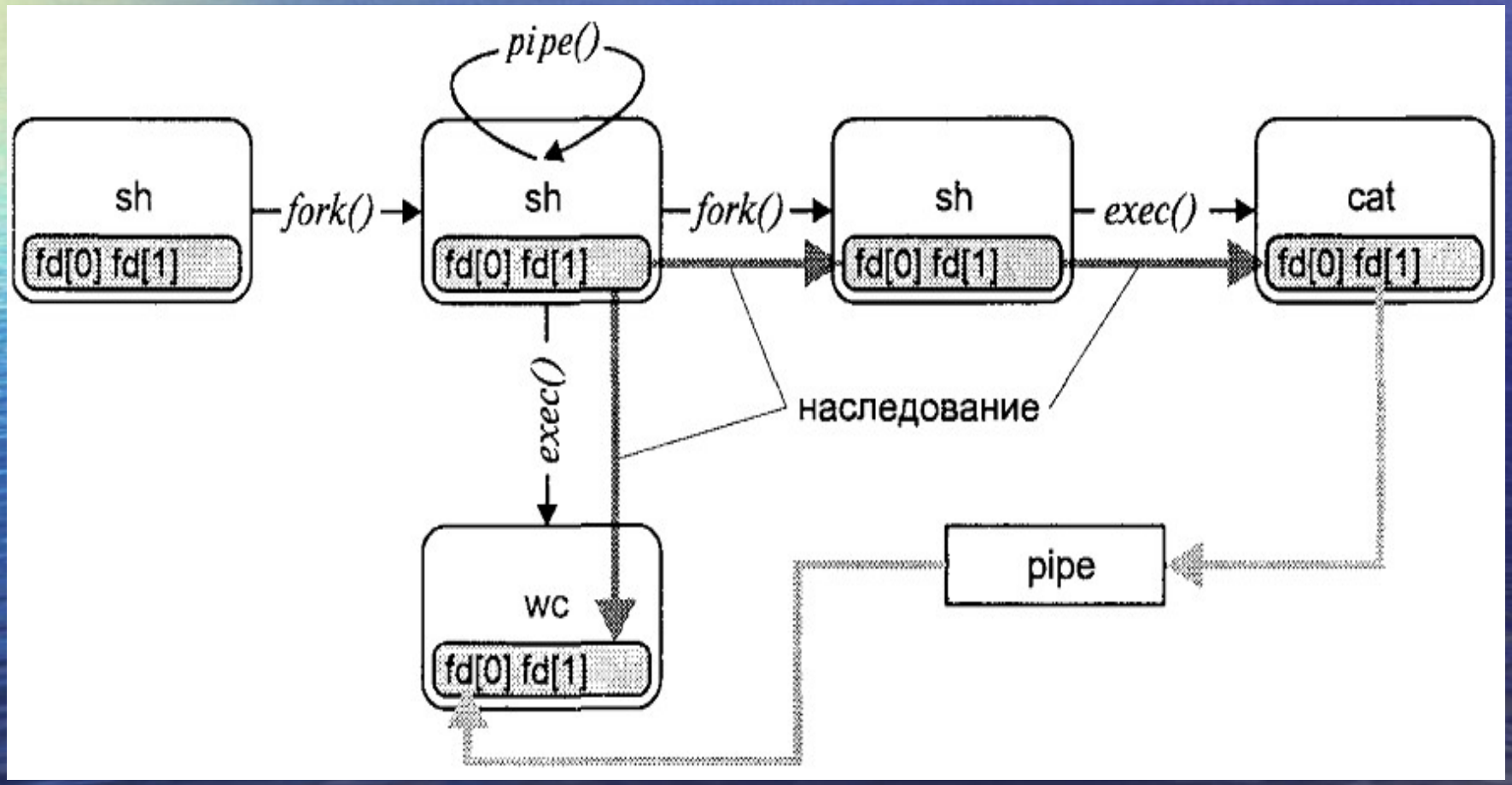
Удаление файла типа FIFO не влияет на уже работающие с данным каналом процессы.

# Конвейер команд

Командный интерпретатор shell использует механизм неименованных каналов при создании конвейера команд:

$ cat myfile | wc

# Создание канала в конвейере команд



# Блокировки файлов

# Блокировки файлов

Обязательные
(mandatory lock)

Рекомендательные
(advisory lock)

# Блокировки файлов

Разделяемые
(чтение)

Монопольные
(чтение и запись)

# Поля структуры flock

- short l_type – тип блокировки

F_RDLCK, F_WRLCK, F_UNLCK

- short l_whence – точка отсчета, как в lseek()
- off_t l_start – начало записи
- off_t l_len – длина записи
- pid_t l_pid – pid процесса

# Системный вызов fcntl

```
int fcntl(int fd, int flag, struct
flock          * lock);
```

flag: F_SETLK, F_SETLKW, F_GETLK

# Подсистема ввода/вывода

# Драйверы устройств

- Символьные драйверы
- Блочные драйверы
- Драйверы низкого уровня

# Символьные драйверы

Устройства с побайтовым обменом данными: модемы, терминалы, принтеры, мышь и т.п.

Отсутствует буферный кэш.

У терминалов специальная буферизация. (ниже)

# Блочные драйверы

Обмен данными с устройством фиксированными порциями (блоками). Диски.

Используется буферный кэш.

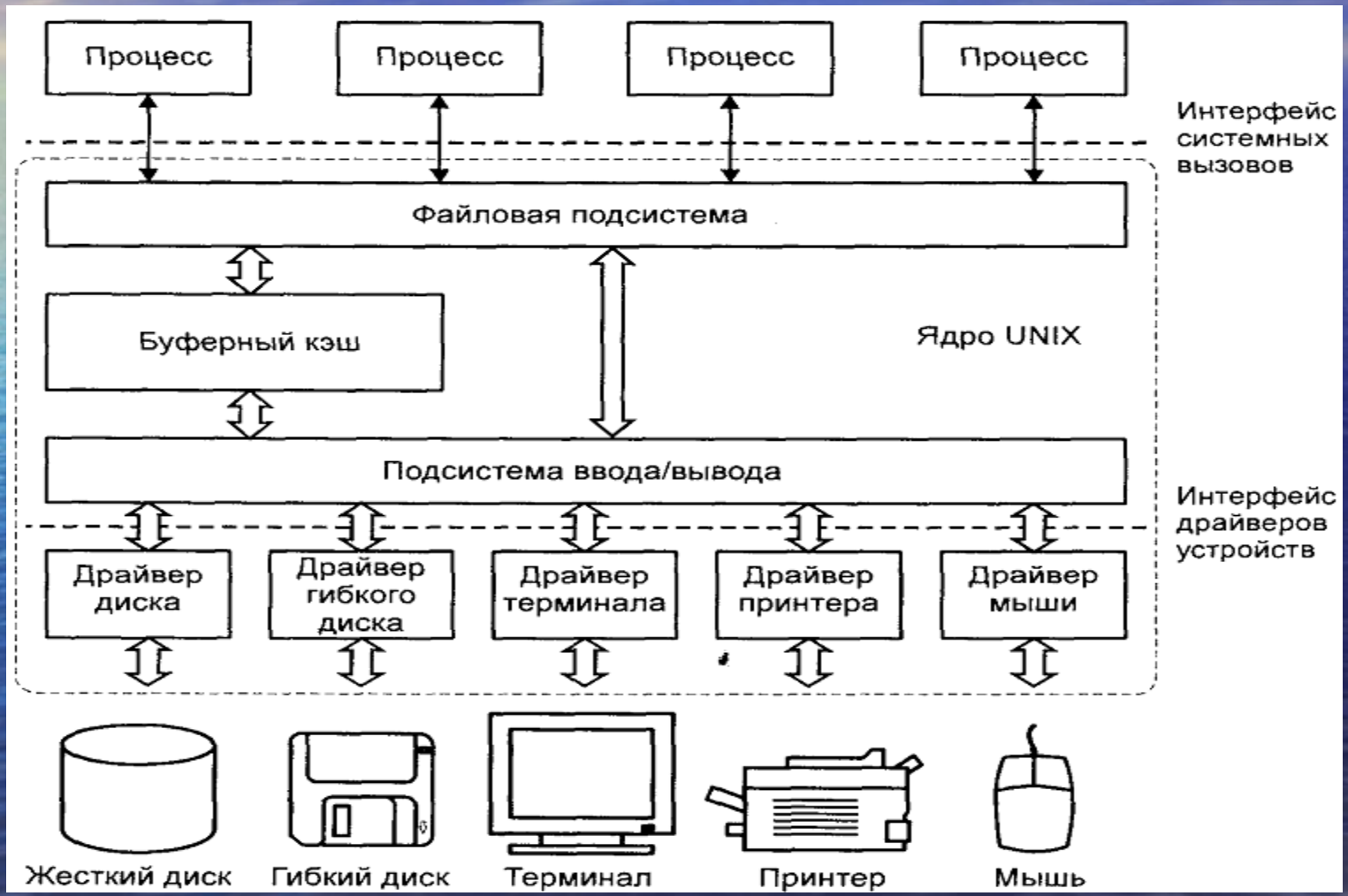
На данных устройствах можно создавать файловые системы.

# Драйверы низкого уровня

Второй интерфейс блочных драйверов для обмена данными с устройством в обход буферного кэша.

Обмен может осуществляться порциями, не равными размеру блока, или побайтово.

# Драйверы устройств UNIX



# Программные драйверы

/dev/kmem

/dev/ksyms

/dev/mem

/dev/null

/dev/zero

/dev/full

/dev/random

# Архитектура драйверов

Старший номер (major number) адресует драйвер.

Младший номер (minor number) передается в качестве параметра драйверу и им интерпретируется.

# Доступ к драйверу

Коммутатор устройств – структура, содержащая указатели на точки входа (функции) драйвера.

Два коммутатора – для символьных и блочных драйверов.

# Элемент массива коммутатора устройств (1)

```
struct bdevsw[]{ /* блочный драйвер*/
int (*d_open)();
int (*d_close)();
int (*d_strategy)();
int (*d_size)();
int (*d_xhalt)();
…
} bdevsw[];
```

# Элемент массива коммутатора устройств (2)

```
struct bdevsw[]{ /* символьный драйвер*/
int (*d_open)();
int (*d_close)();
int (*d_read)();
int (*d_write)();
int (*d_ioctl)();
…
} bdevsw[];
```

# Вызов функции драйвера

(*bdevsw[major].d_open) (major, minor, …);

major и minor могут быть получены с помощью специального макроса из пременной типа dev_t

# Драйвер как набор функций

Не все функции поддерживает каждый драйвер. В этом случае используется заглушка.

Все функции имеют двухсимвольный префикс, например mmopen() – функция open() драйвера kmem.

# Некоторые функции (1)

xxopen() – все типы драйверов.

Реинициализация физического устройства и внутренних данных драйвера. Например, размещение дополнительных буферов.

# Некоторые функции (2)

xxclose() – все типы драйверов.

Вызывается, когда число ссылок на драйвер становится равным нулю. Может вызывать физическое отключение устройства.

# Некоторые функции (3)

xxread(), xxwrite() – не поддерживается блочными драйверами.

Обеспечивает операции чтения и записи с устройством.

# Некоторые функции (4)

xxioctl() – не поддерживается блочными драйверами.

Обеспечивает общий интерфейс управления устройством. Может определять набор команд управления, передаваемых в драйвер с помощью системного вызова ioctl().

# Некоторые функции (5)

xxintr() – все типы драйверов.

Вызывается при поступлении прерывания от устройства. Асинхронная функция для устройств, обслуживаемых по прерыванию.

# Некоторые функции (6)

xxpoll() – все типы драйверов. Производит опрос устройства на предмет наличия новых данных для чтения или готовности приема данных для записи. Асинхронная функция для устройств, не поддерживающих прерывания.

# Некоторые функции (7)

xxhalt() – все типы драйверов.

Останов драйвера при останове системы или выгрузке драйвера.

# Некоторые функции (8)

xxstrategy() – не поддерживается символьными драйверами.

Обеспечивает операции блочного ввода/вывода. Может обеспечивать собственную стратегию выполнения операций целью оптимизации.

# Некоторые функции (9)

xxprint() – все типы драйверов.

Выводит сообщения драйвера на консоль при загрузке или изменении состояния драйвера.

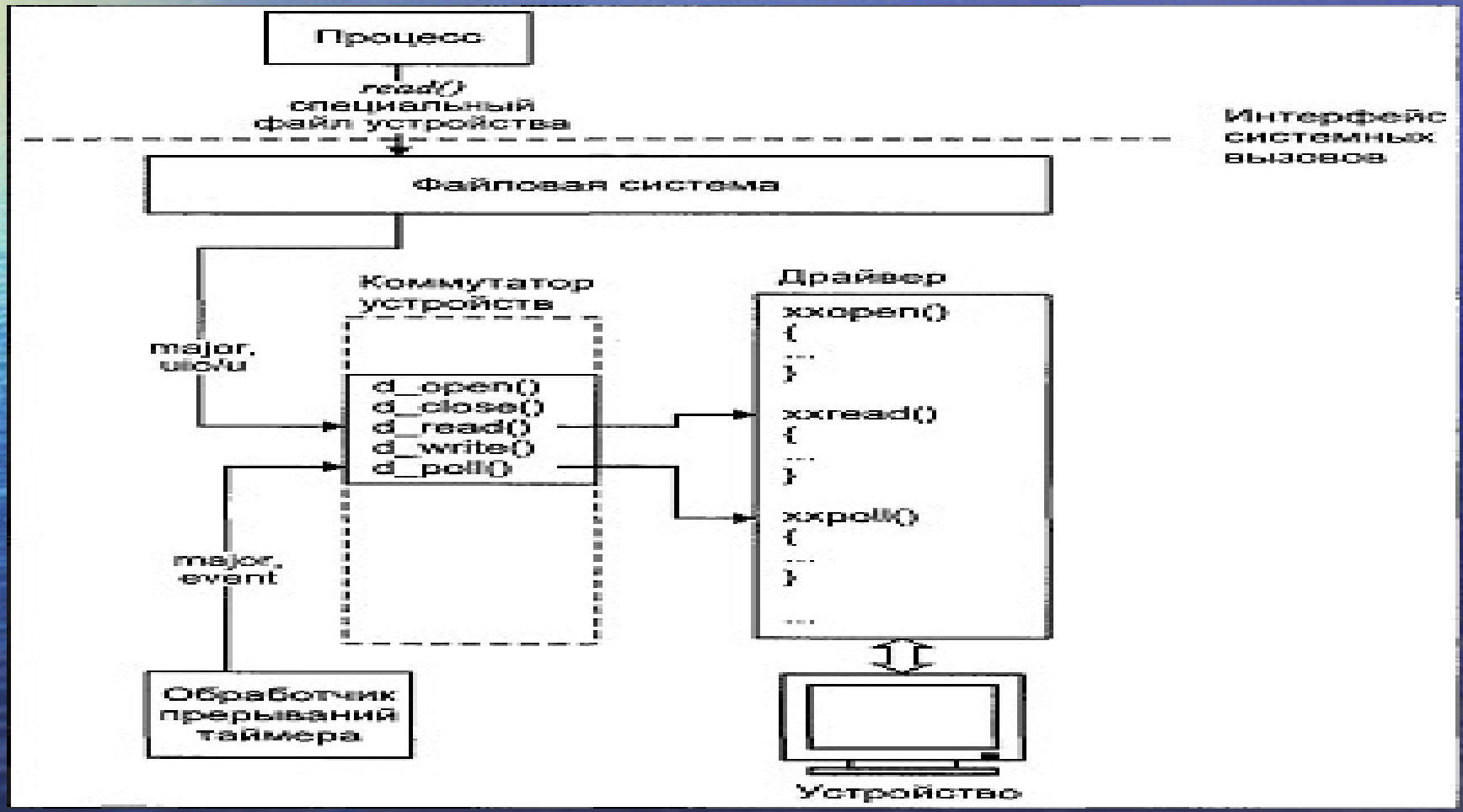
# Обращение к драйверу

- Автоконфигурация
- Ввод/вывод
- Обработка прерываний
- Специальные запросы ioctl()
- Реинициализация/останов

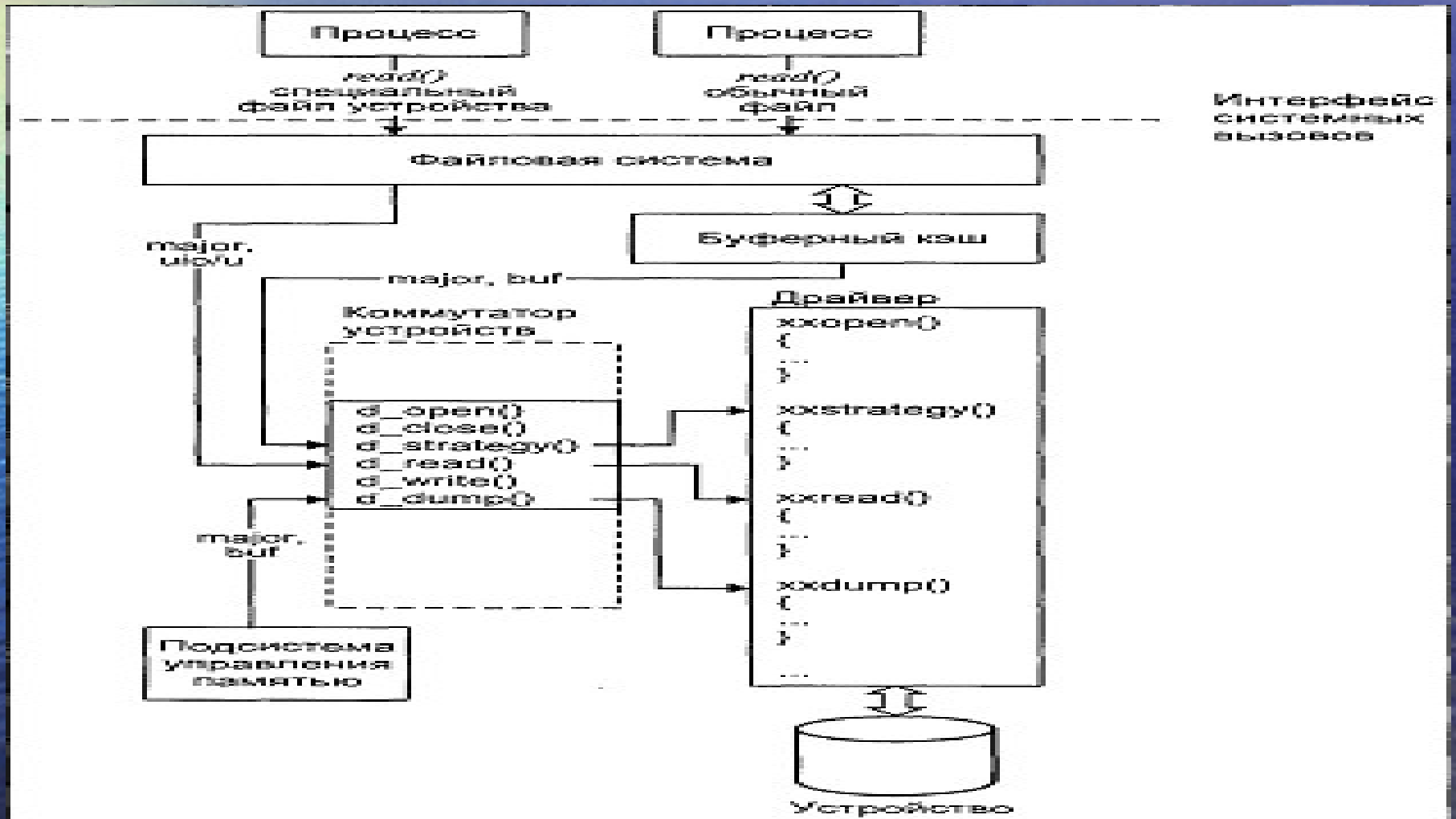
# Три контекста вызова

- Контекст задачи (по запросу процесса)
- Системный контекст (по запросу подсистемы ядра, например, страничный демон)
- Контекст прерывания (обработчик прерывания)

# Доступ к драйверу символьного устройства



# Доступ к драйверу блочного устройства



# Синхронная и асинхронная части драйвера

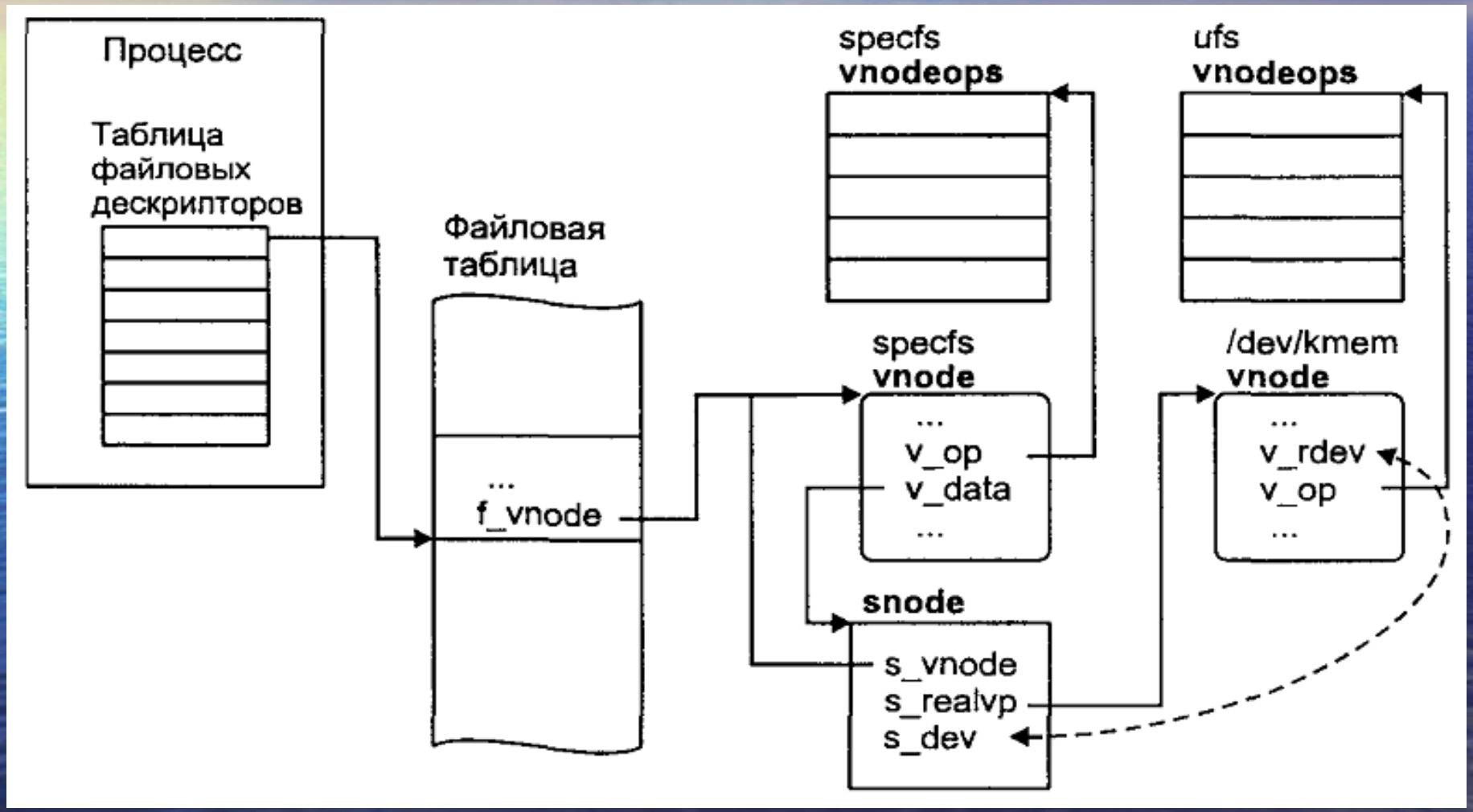
Синхронная (верхняя часть) – по запросу процесса в контексте процесса (доступна u-area).

Асинхронная часть (нижняя часть) – по запросу ядра (прерывания) в другом контексте.

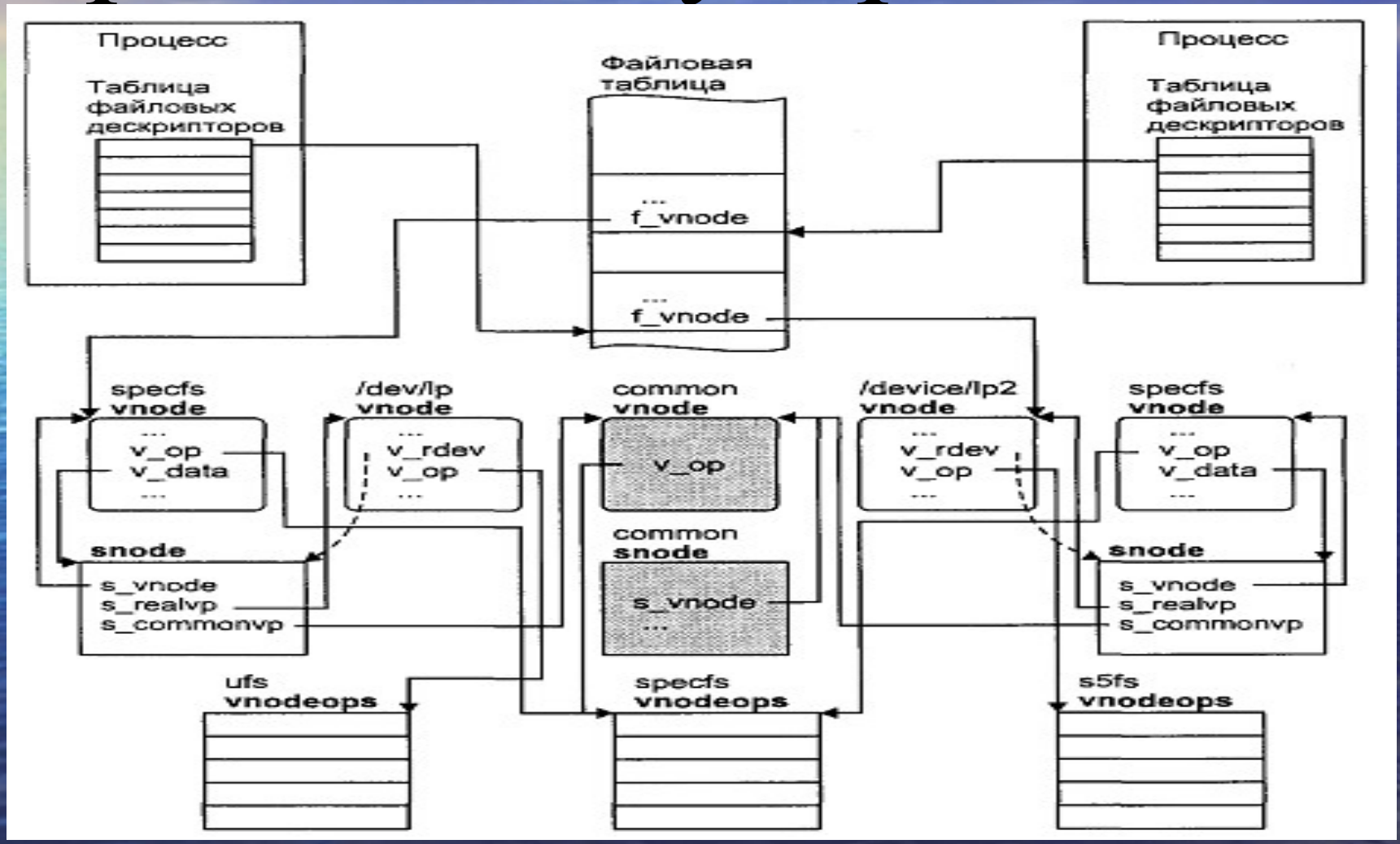
# Файловый интерфейс

Используется специальный тип файловой системы devfs (specfs). Часть функций реализуется vnode файловой системы, содержащей файл устройства, а часть – vnode специальной файловой системы.

# Работа с простым драйвером



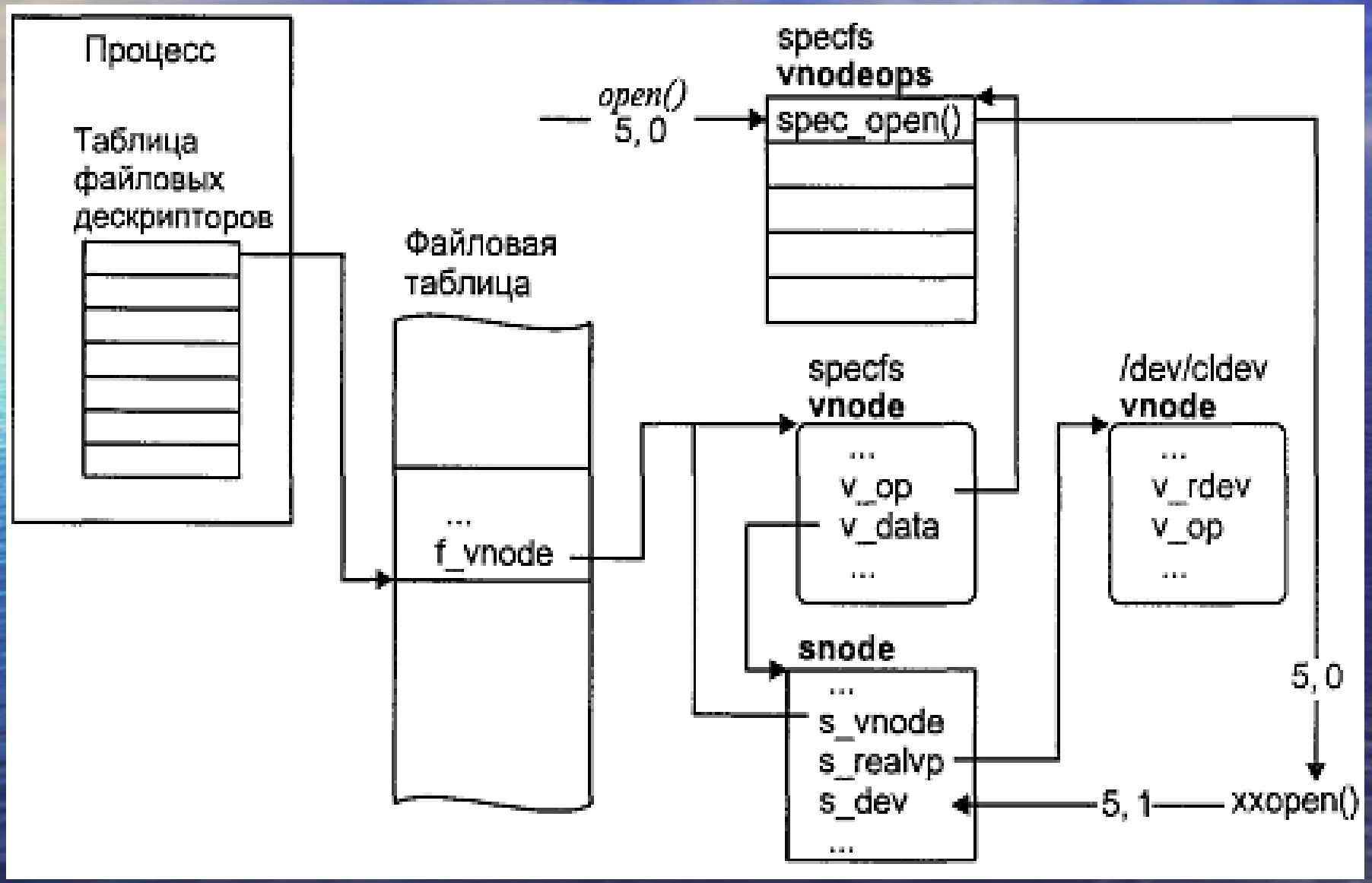
# Работа с драйвером реального устройства



# Клоны

Специальные устройства, работа с которыми требует создания структур данных для каждого процесса: псевдотерминалы (pty), сетевые протоколы (ip, icmp, arp, le, tcp, udp). Младший номер генерируется автоматически.

# Работа с клонами



# Связь драйвера с ядром

- Статические драйверы
- Динамические драйверы

# Динамическая установка драйвера (1)

- Размещение и динамическое связывание символов драйвера
- Инициализация драйвера и устройства
- Добавление точек входа драйвера в коммутатор устройств

# Динамическая установка драйвера (2)

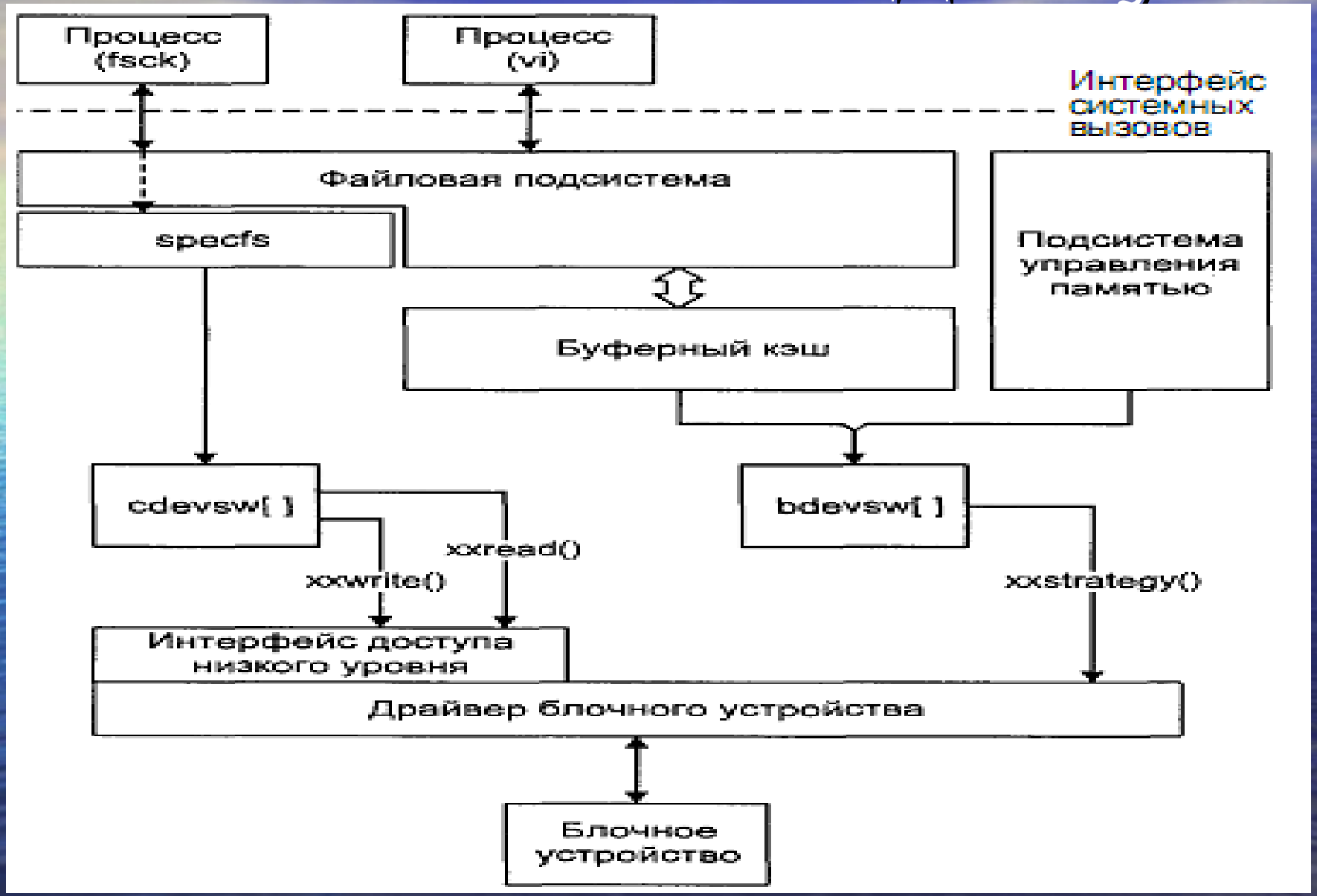
- Установка обработчика прерываний драйвера

# Доступ к блочным драйверам

Два интерфейса:

- С использованием буферного кэша
- Напрямую к устройству без буферизации

# Различные типы доступа



# Буферизация при работе с символьным драйвером

Если устройство поддерживает прерывания, то используется функция xxintr()

Если устройство не поддерживает прерывания, то – xxpoll()

# Архитектура терминального доступа

Последовательный интерфейс, называемый терминальной линией.

Дисциплина линии – предварительная обработка данных после ввода или перед выводом.

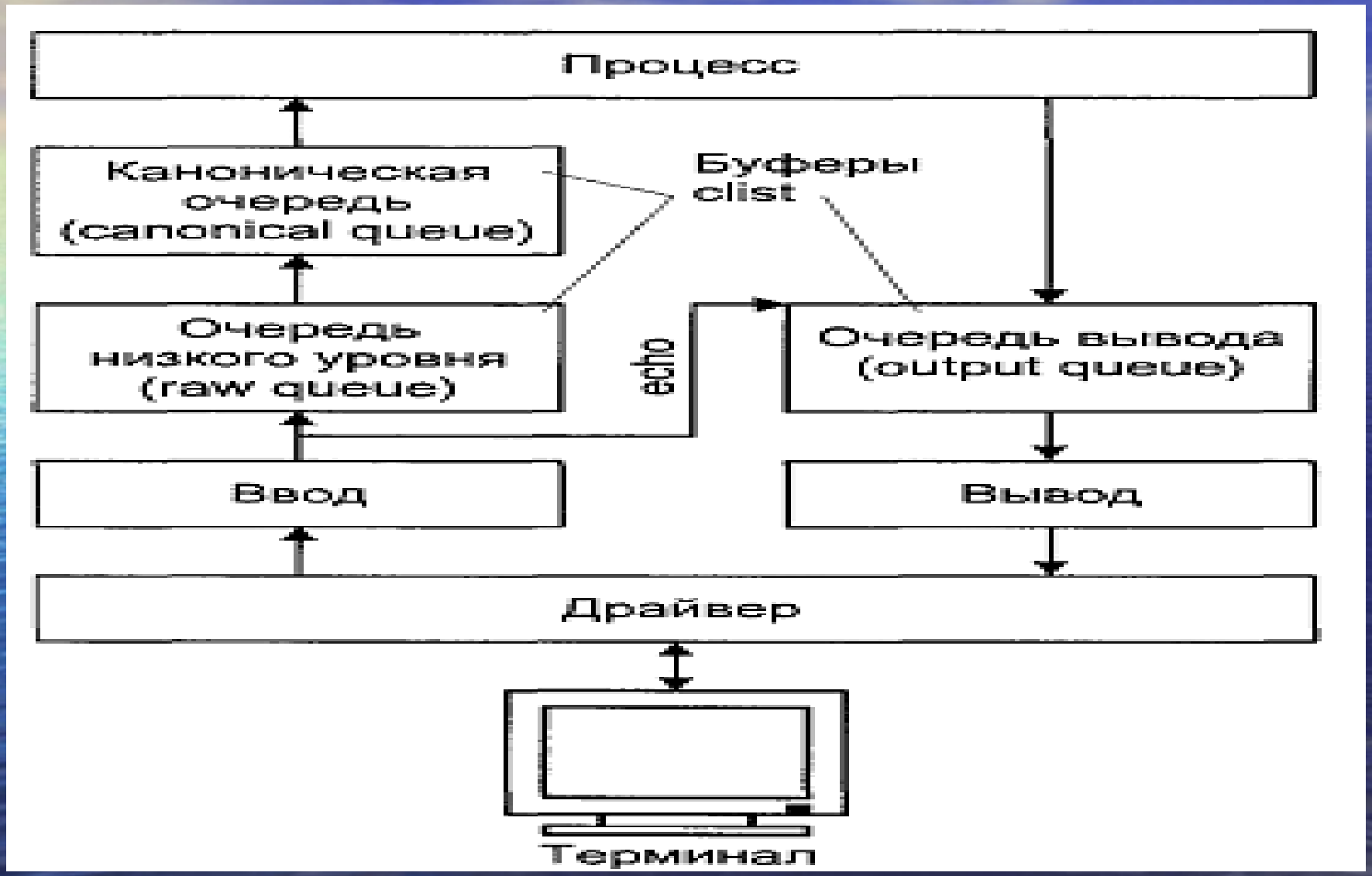
# Два режима работы

- Канонический (в виде законченных строк с использованием дисциплины линии)
- Прозрачный (напрямую с устройством)

# Дисциплина линии

- Построчный разбор введенных последовательностей
- Обработка символов стирания и удаления всего ввода
- Отображение вводимых символов (эхо)
- Расширение вывода (табуляция)

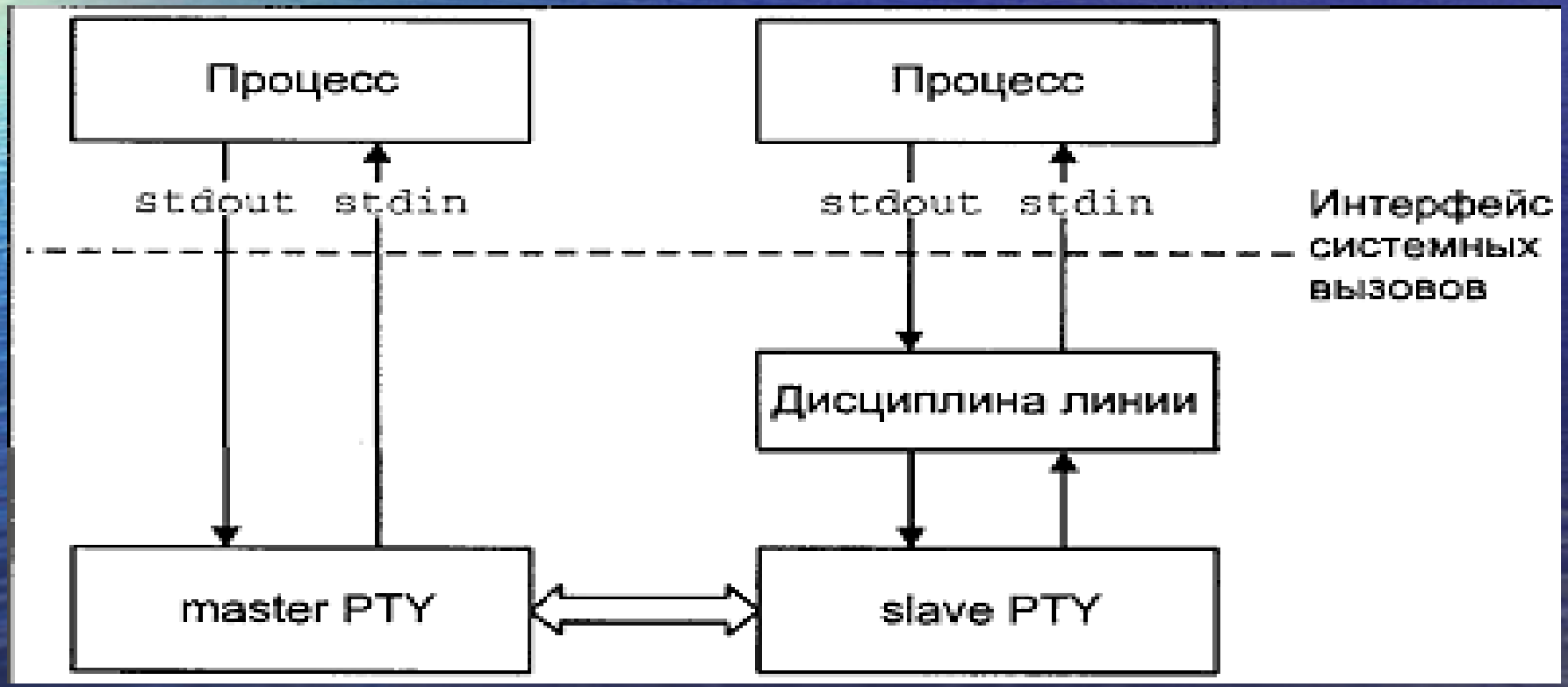
# Работа драйвера терминала



# Псевдотерминалы

Эмулятор терминала. Драйвер состоит из двух частей: обычный терминальный драйвер (подчиненный slave) и управляющий (основной master).

# Взаимодействие процессов с использованием псевдотерминала



# Удаленный доступ